Conceptual Architecture Document

Lori Gildersleeve  
Cody Nelson  
Jennifer Martin

**Conceptual Architecture for a Simple Sudoku Game**

**Overview of Components and Connectors**

The architecture for the Sudoku game will be divided into three main components:

1. the user interface
2. the rendering/verification component
3. the data model/data store

Each of these three components will have their own type of connectors to match:

1. event connectors to handle user input
2. procedure connectors to handle logic
3. data access connectors to handle state/model requests

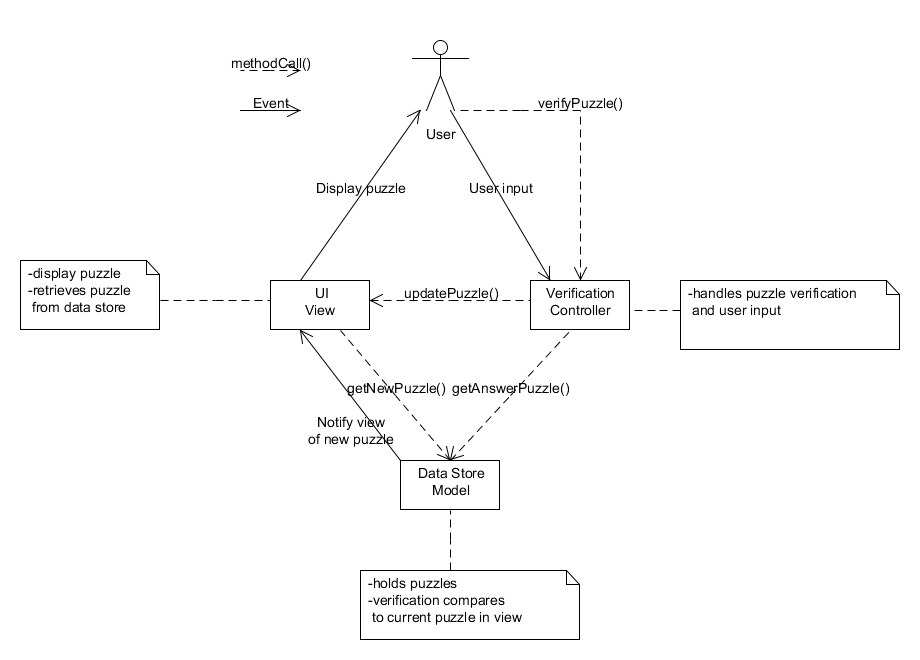
The user interface will display a graphical representation of the Sudoku puzzle. It will respond to user manipulation via event connectors. A user will select an available square and type a number to fill in that square. When a user wishes to validate an input or set of inputs, the user will click the Verify/Solve button.

The render/verification component will handle the logic to both render the puzzle for display and check if each user move is viable to complete the puzzle. It will use procedure connectors, in particular method calls. After the user presses the Verify button, the verification component will check the correctness of the user’s input. Each square will change its presentation if its input is incorrect.

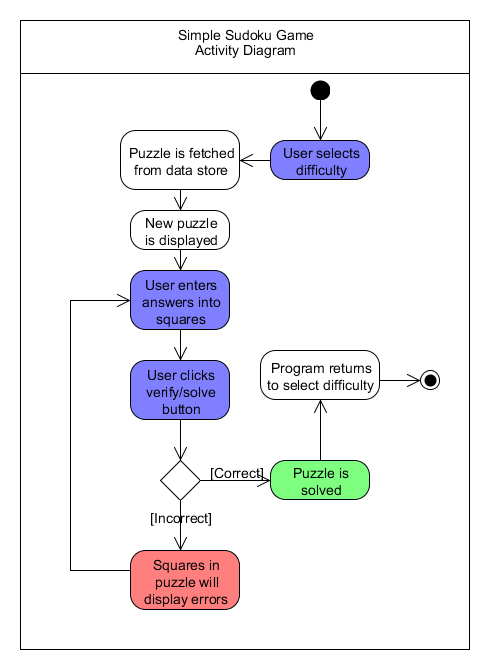
The data store will contain the information representing the Sudoku puzzle. It will be changed with data access connectors. This component will hold the various types of available Sudoku puzzles, from easy to difficult.

These components will use a model-view-controller pattern to interact with one another. Provided is a simple diagram (**Figure 1**) showing how the user will interact with our Sudoku program and how these three components will exchange interactions with one another.

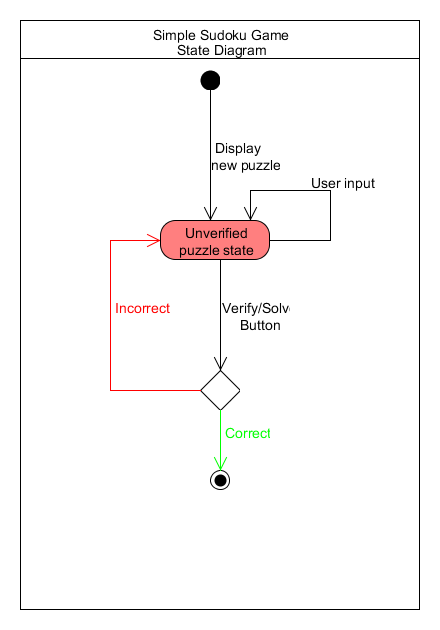
Also provided are an activity diagram (**Figure 2**) and a state diagram (**Figure 3**) to show how the user will interact with the Sudoku program.



**Figure 1**: Simple MVC diagram for Sudoku program



**Figure 2**: Activity diagram for Sudoku program



**Figure 3**: State diagram for Sudoku program

**Basic Game Design Principles**

**Simplification for Testing**

The Sudoku game is designed to be played by a single user on a single machine. These design decisions make it simple and easy to test the program. The developers can deploy the code base to one machine and test it themselves as the sole user.

**Language Influences Portability**

The game will be built using Java 1.7, because it is an object-oriented language that a majority of the team members are familiar with using. Because the program will be written in Java, it can be played on machines with any operating system that have a viable Java Virtual Machine implementation. This code portability is a side effect of the language choice.

**Crosscutting Concerns**

Validation and sanitization will occur throughout the entire application. Therefore, methods being utilized by more than one area will be housed centrally for efficiency and ease of management. Input will be validated by what is deemed acceptable, thereby making it easier in the future to expand upon acceptable inputs, as opposed to attempting to narrow unacceptable inputs. This will aid in adaptability in the future.

Logging and Exception Handling will be handled by Java and Groovy libraries for consistency and robustness of functionality.

The state of the Sudoku application will be kept lean in order to optimize performance and memory management. The absolute minimum amount of necessary data to be persisted will be stored in the state.

**Conceptual Integrity**

Additional design ideas will be considered in depth before implementation. If the idea is deemed a good fit for the Sudoku architecture, the architecture will be updated and analyzed prior to any development. This will aid in ensuring the conceptual integrity of the Sudoku application.