# SOFTWARE DESIGN DOCUMENT

Author: Ruslan Afanasiev
Date:   25 September 2011

## REVISION SUMMARY

|  |  |  |
|---|---|---|
| 1. | Date: | Sep 25, 2011 |
|  | Revision: | 0.01 |
|  | Changes: | Initial version |
| 2. | Date: | Sep 29, 2011 |
|  | Revsion: | 0.02 |
|  | Changes: | Flow chart diagram |

**TABLE OF CONTENTS**

1. Introduction
2. Design overview
3. Data design
4. User interface design
5. Component and procedural design

## PREFACE

This document provides an overview of the design for the Movie DB software system.

## 1. INTRODUCTION

The Movie DB software allow an user to manage own movie's catalogue:

- add a new record about favorite movie
- view current record by ID
- search movie by title or movie star's name
- delete a record
- import a data from flat text files

## 2. DESIGN OVERVIEW

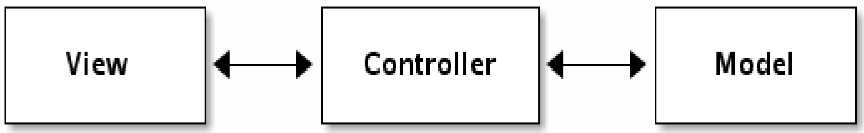


**Figure 1** - Relation between application components

As a main software design pattern, in this case, has been decided to be used an architectural pattern Model-View-Controller (MVC). The main idea was following - divide an user interface, an application logic and a data storage system. The controller interacts between the user interface and the storage system to convert event into an action to manage the data via the model.

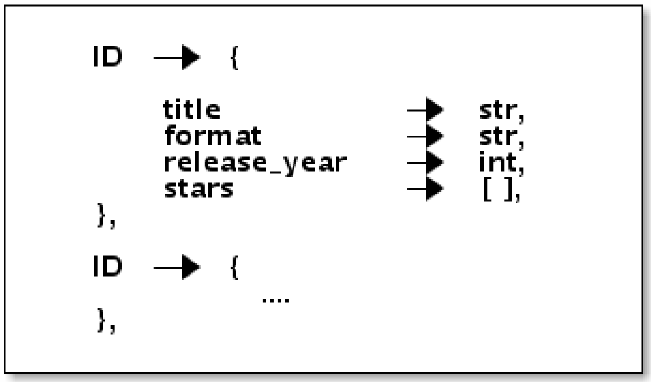
The structure of components:

- controller class realized as a Perl module - **Library.pm**
- model class realized as a Perl module - **Storage.pm**
- view class realized as a Perl module - **Prompt.pm**

## 3. DATA DESIGN

The data storage contains 'raw movie objects' to simplify data manipulation actions. It gives us to manipulate the data without any kind of extra transformation. Every data item is the object. We can change object's attributes using couple of methods. This solution has been provided by a very common package in Perl - the module **Storable.pm**. This module allows us serializing or deserializing complex data structures into the file system.

The single movie object is stored as:



```
ID  →  {
     title           →  str,
     format          →  str,
     release_year    →  int,
     stars           →  [ ],
},
ID  →  {
          ....
},
```

**Figure 2** - Representation of the data structure

| | |
|---|---|
| • ID | - unique numeric descriptor |
| • title | - name of a movie (string) |
| • format | - could be VHS, DVD or Blue-Ray (string) |
| • release_year | - year of movie's production (digit) |
| • stars | - array reference with all stars in the movie |

**TODO**: migrate to more powerful data storage system (RDBMS, e.g. SQLite)

## 4. USER INTERFACE DESIGN

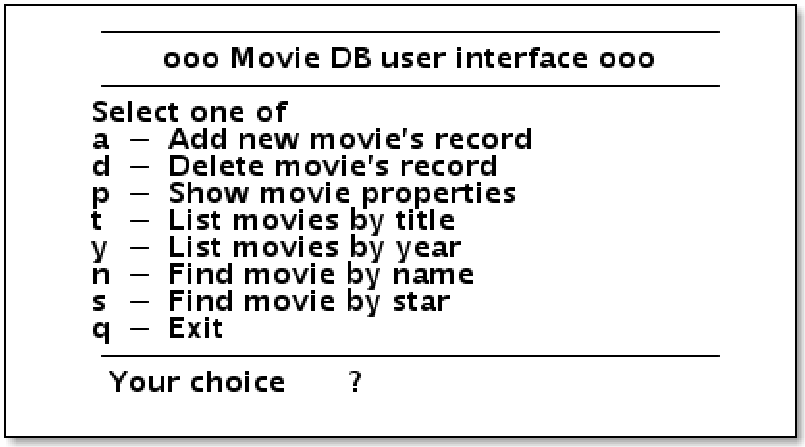
The application provides two ways to operate with the data:

- a command-line interface (CLI)
- a text user interface (TUI)

The first one (CLI) is used in a common UNIX style and gives an user several keys to pass them to a script (movie_storage.pl) to manipulate the data. Here are command-line options:

| | |
|---|---|
| -i, --import | import movies from a data source txt file |
| -a, --add | add a movie's record. The argument string should looks like: *./movie_storage.pl --add 'release_year=YYYY;title=The movie;format=DVD;stars=John Doe'* **NOTE**: the argument string should be quoted! |
| -d, --delete | delete a movie (it will ask movie ID) |
| -p, --property | show properties of a movie (it will ask movie ID) |
| -t,  --title-list | show list title and IDs of all movies, alphabetically |
| -y, --year-list | show list title and IDs of all movies, chronologically |
| --find-by-name | find movies by name (it will ask for a search string) |
| --find-by-star | find movies by star (it will ask for a star name) |
| -m, --menu | script will prompt an interactive menu to add a new record, etc |

The second (TUI) provides a menu to select options via an interactive interface.



```
ooo Movie DB user interface ooo

Select one of
a  –  Add new movie's record
d  –  Delete movie's record
p  –  Show movie properties
t  –  List movies by title
y  –  List movies by year
n  –  Find movie by name
s  –  Find movie by star
q  –  Exit

 Your choice      ?
```

**Figure 3** - sample of the menu session

**TODO**: create more mature GUI or WEB interface (e.g. Perl/Tk or CGI script)

## 5. COMPONENT AND PROCEDURAL DESIGN

Consider following diagram to track down how each component interacts to others:

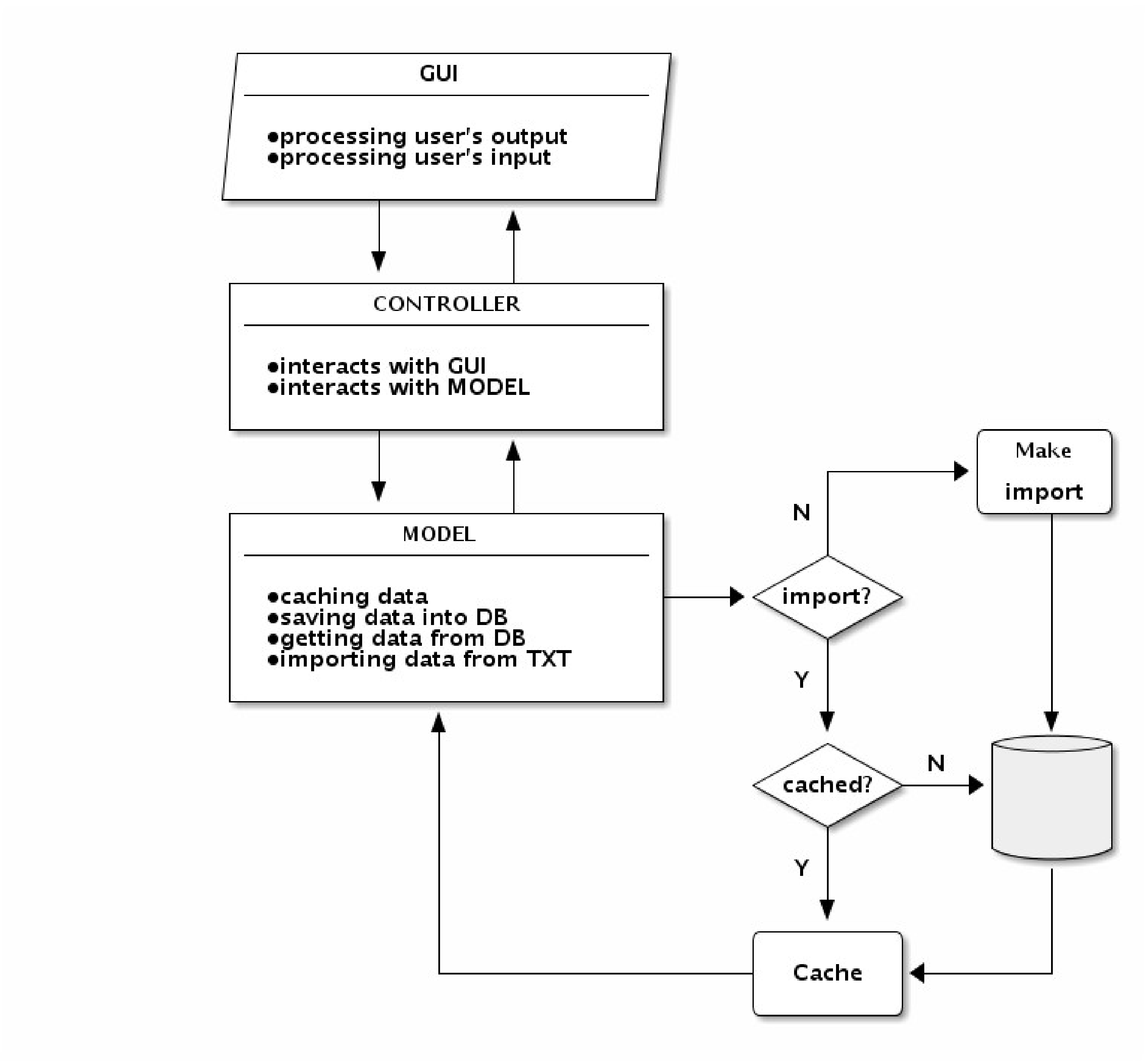


**Figure 4 –** the work flow diagram

Consider class's diagrams to describe the structures of each class, at least all fields and methods:

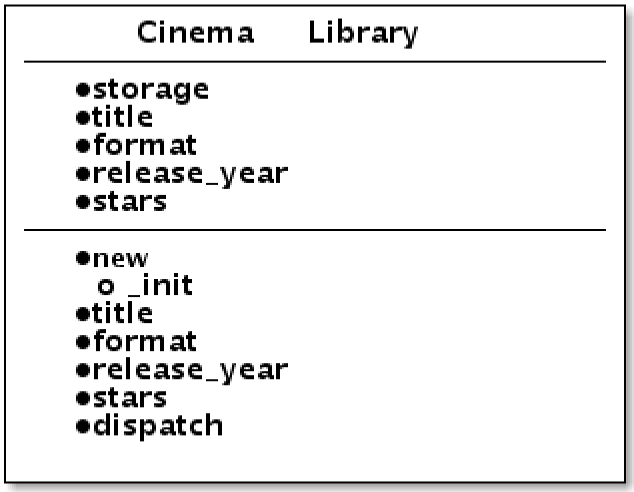


**Figure 5** – Class Cinema::Library

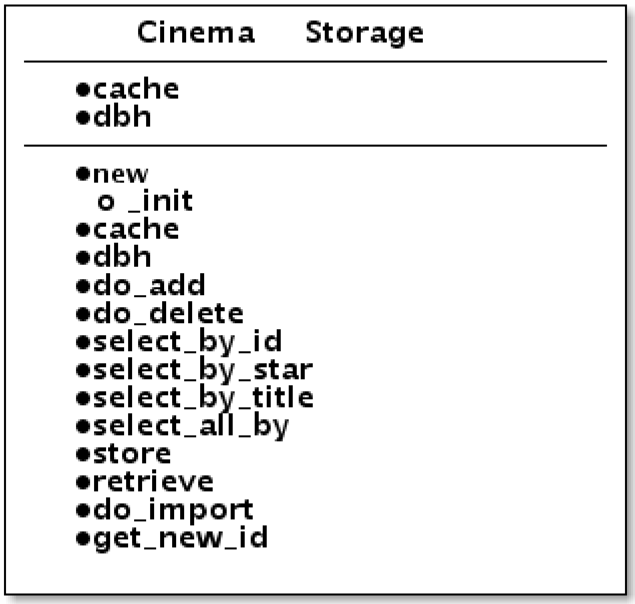


**Figure 6** – Class Cinema::Storage

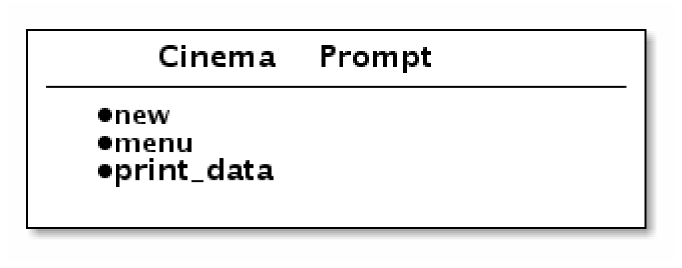


**Figure 6 –** Class Cinema::Prompt