TDT4240

Software Architecture

Spring 2012

Group A17
Android SDK

Steinar Haram
Emil Andreas Mork
Stian Sørebø
Ole Jørgen Rishoff

Architecture Document v1.2

Primary focus:
Maintainability

Secondary focus:
Usability

Contents

List of Tables

List of Figures

1 Introduction

This course teaches architectural styles and patterns, methods for constructing and evaluating architectures, and comnponent-based development. Design patterns and object-oriented frameworks are also covered.

The goal of this project is to develop a fully functional two player version of the classic board game *Nine Men's Morris*, using the methods and styles corresponding with the course goals.

According to the recommended practice of IEEE 1471, this document details architectural patterns and tactics based on all of the underlying drivers, stakeholders and their concerns. The 4+1 View model is a recommended practice for architecture description of software-intensive systems, and will be used to describe the system from the viewpoint of different stakeholders, such as end-users, developers and project managers.

The second chapter will cover the architectural drivers for this project. The third chapter will cover stakeholders and concerns, while the fourth chapter contains a selection of architectural viewpoints. Architectural tactics and architectural patterns will be covered in chapter five and six, respectively. The seventh chapter contains a selection of views. This includes a logical view with a class diagram, a development view, a process view with an activity diagram, and lastly a physical view showing how our system interacts with a third party server solution. Chapter eight concerns the consistency among our views, and the following chapters cover architectural rationale, issues, document changes and at last references.

2 Architectural drivers

Project selection was based on being able to use the architectures that were used in the course. This is to maximize the learning outcomes and make the project feasible. We also want to create an architecture that can be developed without any big problems. The different architectural drivers, that has shaped the project is listed under in sub-sections.

2.1 Functional requirements

Functional requirements is a big architectural driver because the architecture must adapt to the functional requirements to ensure desired functionality.

2.2 Quality attributes requirements

Having modifiability as our main quality attribute will be a huge architectural driver and will be kept in mind when build the architecture to ensure changeable and modular code.

2.3 Technical constraints

The technical constraints is an architectural driver because the architecture must adapt to the Android platform.

2.4 Business constraints

The project lifetime is only 10 weeks. We must limit the functionality and possibilities of the project to keep it a realistic project.

2.5 Fast start up

The faster we get the architecture done, the faster we can start programming. We will try to be one week a head of the plan. This will make our schedule resistant for problems that might arise.

3 Stakeholders

Stakeholder	**Concern**
Developer	Buildability: The game needs to be finished within a short period or time. Modifiability: The game should be easy to extend and change current features. Testability: It should be easy to test whether the game is working as intended or not. Grade: The project evaluation criteria will be prioritized to achieve the best grade possible.
Player	Usability: It should be easy to both use and learn by new players. Playability: The game should be interesting and fun to play. Easy setup: The knowledge and time needed to install the game should be trivial.
Course staff	Reviewability: The delivered code should be readable, and the setup should be easy.
ATAM evaluator	Reviewability: There should be good architectural documentation that consist with the code.
Android Market	The final application should follow the guidelines for publishing applications on Android Market.

Table 1: Stakeholders

4 Architectural viewpoint selection

The 4+1 View Model consist of four different views: Logical View, Development View, process View and physical View. Our game will be a 2-player network game, but since we are going to use a third party library for the network communication we will not use the Physical View.

4.1 Logical view

Basis	We will put the system into perspective and get an overview of the software architecture. Standard procedure is to divide the program into different object-oriented models and find their relationship.
Stakeholders	Developers, course teachers
Description	Standard Class diagram with package notation

Table 2: Logical view

4.2 Development view

Basis	Used to get a perspective view of the different main blocks of the systems.
Stakeholders	Course teachers, Developers
Description	Layer diagram

Table 3: Development view

4.3 Process view

Basis	We use the process view to get an overview of the process flow. This can help the developers to design and understand the main logic and structure of the game.
Stakeholders	Developers, Course teachers
Description	Activity/Sequence diagram

Table 4: Process view

4.4 Physical view

Basis	We use the physical view to get an on overview of software components on the physical layer, and communication between these components.
Stakeholders	Developers, Course teachers
Description	Deployment diagram

Table 5: Pysical view

5 Architectural tactics

5.1 Modifiability

5.1.1 Prevention of Ripple Effect

The code should contain as few dependencies as possible in order to avoid ripple effect when making changes to the code.

5.1.2 Localize Changes

As our quality requirements highlight in the Requirements Document [1], we have anticipated several possible extensions to our game application. We have taken this into account, and will build our architecture with this in mind.

5.2 Usability

5.2.1 Design-time tactics

- The GUI will be separated from the model, thus making it easy to carry out changes that only concerns the graphical layout.
- The system will use a third party server solution for the multiplayer feature, which makes the job easier for the developers.

5.2.2 Runtime tactics

- The system shall, as mentioned in the functional requirements, highlight the user's possible moves.
- The system shall inform the user whose turn it is during gameplay.

5.2.3 Others

- To optimize the usability of our user interface, we will get feedback from outside users during the project.
- We will follow the Android user interface guidelines. These design principles were developed by and for the Android User Experience Team to keep users' best interests in mind. We will consider them as we apply your own creativity and design thinking.

6 Patterns

6.1 Architectural patterns

6.1.1 MVC

Model-View-Controller (MVC) is an architectural pattern that divides interactions between users and applications into three roles: the Model (business logic), the View (user interface), and the Controller (user input). This separation makes is possible to develop, test and maintain each role independently.

MVC pattern will be used when implementing *Nine Men's Morris*, where we will isolate the domain logic from the user interface. In Android the activities can act as view and controller. We will need to create a MVC logic that is separated from the activities, with a controller that handles player interactions, models that executes the input from the controller, and a view that gets updated via the interfaces.

6.2 Design pattern

6.2.1 States

The state pattern is used to represent the state of an object. This a clean way for an object to partially change its type at runtime.

The state pattern will be implemented in order to ensure high modifiability, and to be able to smoothly change the state of the game. The different states of the game is shown in the logical view in section 7.1.

6.2.2 Singleton

The singleton pattern is used to implement the mathematical concept of a singleton, by restricting the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system.

To avoid unnecessary instantiation of objects, we will implement the singleton pattern where we see fit. This will benefit the developer that only needs to deal with one instance of the object in question.

7 Views

7.1 Logical view

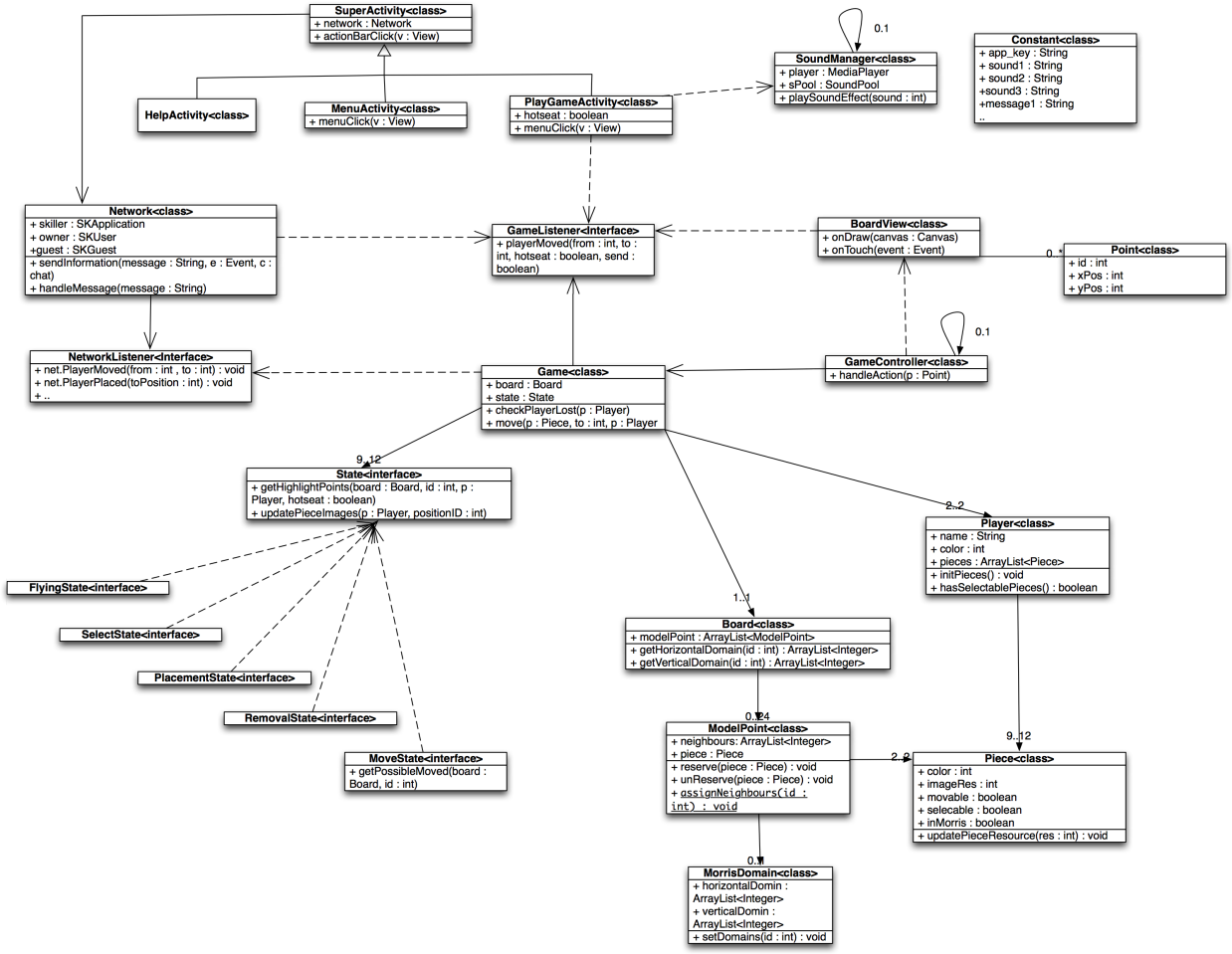

Figure 1: Logical view

The diagram follows the 4+1 logic view notation suggested by the Kructhen article [2]. The class diagram shows the structure of a system by showing the system's classes and the relationships among them. Aggregation and inheritance is also displayed.

7.2 Development view

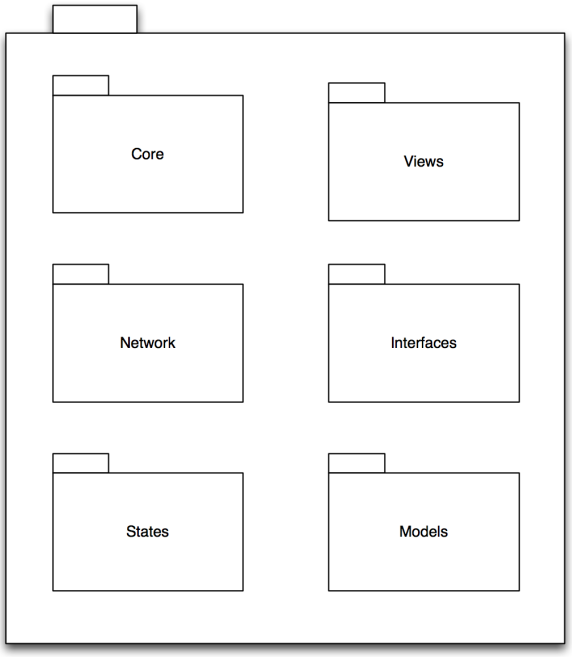

Figure 2: Development view

The development view shows how we have organized our project. The diagram shows a rough overview of how the packages are organized. We have separated our models, views and activities (within Core) in different packages for a better overview.

7.3 Process view

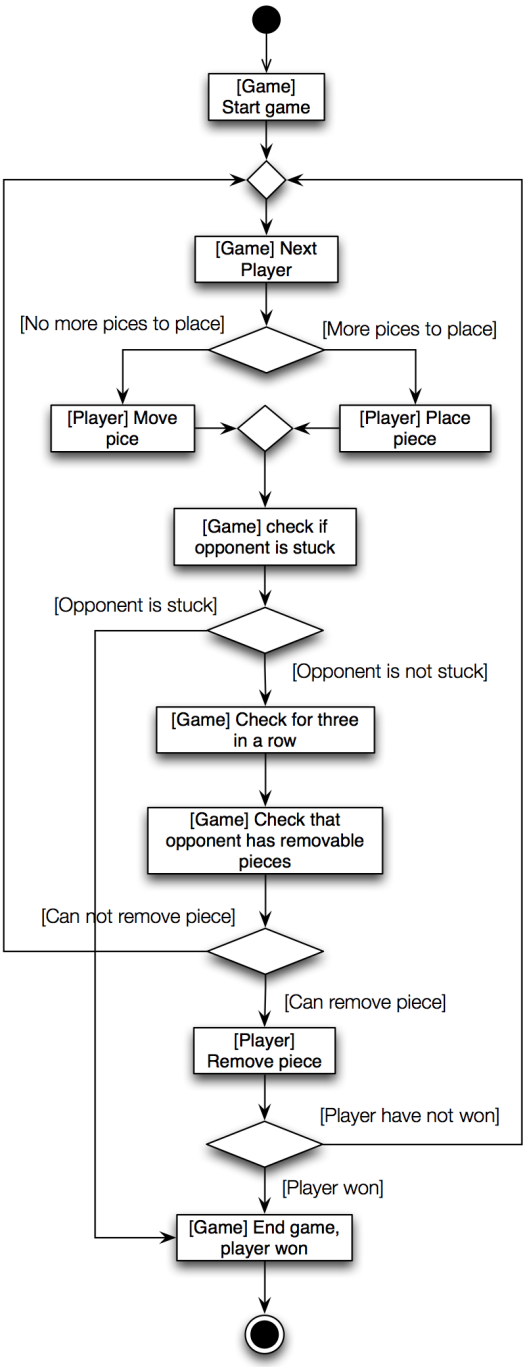

Figure 3: Process view

The game is a turn based game with two players. In the activity diagram the progress of a game is described. When a game is started one of the players starts placing one of its pieces. The game checks if the player get three in a row. In that case the player can remove one of the opponents pieces. If the opponents have less then three pieces left, the other player have won. The diagram shows how a game round will pan out.

7.4 Physical view

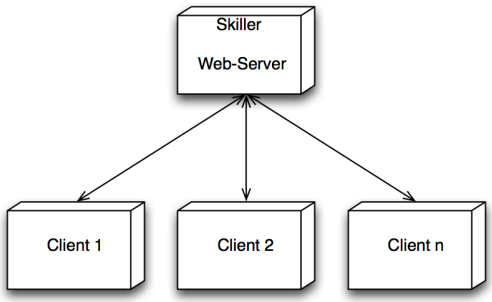

Figure 4: Physical view

The physical view shows how the clients interact with the server of the Skiller framework. This is included for visualization only.

8 Consistency among views

The logical view shows a class diagram with tags indicating the different entity types. The classes that are within the same package in the development view, are placed together. The entity types and class naming conforms with the diagram for the development view, and the process view uses the same class names as shown in the logical view.

The physical view is inconsistent with the other views, and is included to show that the application interacts with the server of the Skiller framework.

9 Architectural rationale

We have chosen to build our architecture with implementation of the MVC, State, and Singleton pattern. The MVC pattern will help us cleanly separate domain logic from the user interface, which conforms with both modifiability and usability. The state pattern allows for extension of the game in a convenient matter, conforming with our main focus. The singleton pattern is used to prevent unnecessary instantiation of objects central to the application.

Usability is extremely important in a gaming environment, and we have therefore chosen to have this as our second area of focus. The end user will have a lot to say in our final design, and we will strive to satisfy this user group in a best possible manner. High usability for the end user will be achieved through gameplay hints and on-screen messages, as mentioned under architectural tactics in section five.

The developer team's usability is achieved through usage of an easily implemented third party network framework, in addition to separation of the graphical interface from the model.

10 Issues

We have had some difficulty finding a multiplayer framework that suits our exact needs, but we do not see this as something that will hinder our progress significantly.

11 Changes

Date	Change
27.02.12	First version of the document
12.04.12	Rewritten introduction and added runtime and design-time tactics.
13.04.12	Edited stakeholders, architectural drivers, rationale, and architectural patterns.
25.04.12	Updated views and consistency among views.

Table 6: Document changes

References

[1] S. Mork, Haram and Rishoff, Requirement Document.

[2] P. Krutchen, The 4+1 View Model of Software Architecture, http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf, 1995.