Anagrams Architectural Design Document

Irritable Enterprises, Inc.

13 June 2008 – Version 0.3 – For Review – 5 pages

1 Introduction

This document defines the architecture for *Anagrams*. It divides the product into a number of components that can be developed and verified independently. The architecture should also serve as a guide to future maintenance of the product.

The reader is assumed to be familiar with the terminology introduced in the *Anagrams* User Requirements Document.

Section 2 provides an overview of the product. Section 3 defines the external interfaces. Section 4 explains the rationale behind the chosen design method, and presents the resulting decomposition from several viewpoints. Finally, Secion 5 specifies in detail the components of the architecture.

2 System Overview

The *Anagrams* product is a simple piece of software: it is a single-user stand-alone program to play a word game. Its purpose is mainly to entertain the user and, hence, it is not a critical system.

No preliminary studies or prototyping were done to select the architecture. One alternative architecture was considered, where the rules of the game were put into a separate component to decouple them from the user interface. However, the visual aspects of the user interface were so closely tied to the rules of the game, that it was decided to keep the user interface and the rules in a single component. Otherwise, an extra interface would have to be introduced, and the complexity of this interface would not outweigh the advantages.

3 System Context

In its current form, the *Anagrams* product has only one external interface, viz. to the human user. This is a graphical user interface, presented in the host operating system that runs the *Anagrams* product.

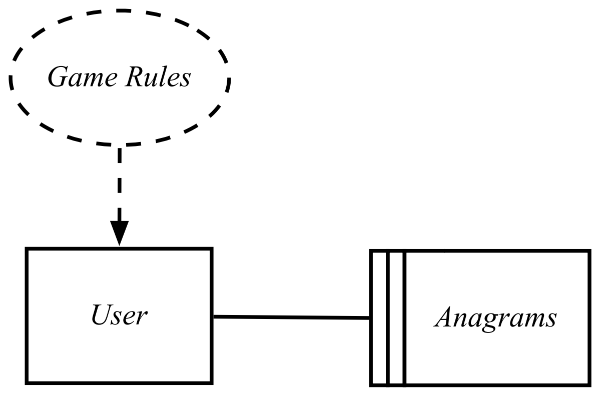

Figure 1: Context diagram for *Anagrams*

Figure 1 shows a context diagram in the form of a *problem frame*. For more information on problem frames see [1, 2]. The box with the triple line on the left is the (abstract) *machine* to be designed, whereas single-line boxes (in this case, only one) denote context *domains*. Edges between boxes indicate *shared phenomena*, that is, an interface relationship. Dashed ovals represent *requirements* to be realized by the machine. The arrow points to a domain to be controlled by the machine in some way.

In the future, an external interface to the file system or even to a network may be introduced. There are two reasons for this:

1. to store the word list separately from the executable, making it possible to change the word list without rebuilding the *Anagrams* product;

2. to store game results for later inspection after the *Anagrams* product terminates execution.

4 System Design

4.1 Design Method

Because *Anagrams* is a very simple system, no specific design method was applied. The design was intuitively obtained through the principle of *separation of concerns*.

4.2 Decomposition Description

Figure 2 shows the decomposition of *Anagrams* into components, and how they relate to each other and to the environment. The dahsed line encloses the *Anagrams* system. Each box inside the dashed line represents a component, and boxes outside the dashed line represent entities in the environment. An arrow from A to B expresses that A *controls* B.

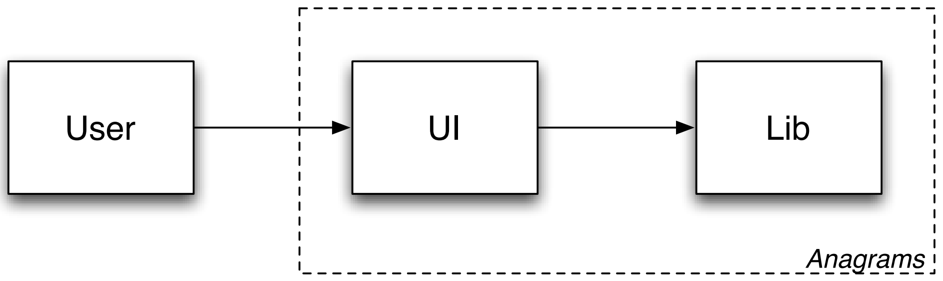

Figure 2: Design-level decomposition diagram for *Anagrams*

This is an abstract (design-level) static (structural) view of the system, consisting of the two components *UI* (User Interface) and *Lib* (Word Library). A more concrete (implementation-level) static view of the system is shown in Figure 3. This is a UML *package diagram*. Each component is implemented as a separate package. The dashed arrow indicates package dependence.

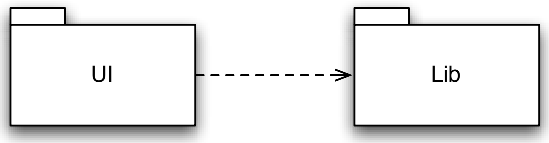

Figure 3: Package diagram (at implementation-level) for *Anagrams*

Because of the simplicity of *Anagrams*, a dynamic view reveals only one process for the entire application, which runs on a single processor. In the future, it may be considered to run the user interface in a separate thread. It also an option to have multiple clients (user interfaces) share the word library by running the latter as a server which is accessed through a network.

5 Component Description

Each component is described in more detail in the following subsections.

5.1 Component UI: User Interface

Type This component is a separate Java package: `com.toy.anagrams.ui`.

Purpose Its purpose is to encapsulate the graphical user interface and those rules of the game that concern the ordering of game-related events. In particular, it realizes user requirements UR-10, UR-11, UR-15[1] UR-16, UR-20, UR-30, UR-40, UR-80, UR-81.

[1]Partially, in that it presents the result to the user.

Function This component presents the graphical user interface, handles all user-generated events, and enforces the rules of the word game.

Subordinates It has no subordinates.

Dependencies It requires **Lib**, the Word Library.

Interfaces This component provides an external graphical user interface and requires the interface to the Word Library.

The graphical user interface has elements

- to present the anagram (scrambled word);
- to let the user type in a guess (UR-20);
- to let the user indicate that a guess is to be evaluated (UR-20);
- to inform the user of the correctness of a guess (UR-15);
- to let the user start a new round with a new word (UR-20);
- to display the game score (UR-16);
- to show the product's version (UR-30).

Resources It requires access to the screen, mouse, and keyboard.

Processing This component handles events dispatched from the main event loop. It realizes the game state transitions and related input and output via the graphical user interface.

Data It maintains the current game state.

5.2 Component Lib: Word Library

Type This component is a separate Java package: *`com.toy.anagrams.lib`*.

Purpose Its purpose is to encapsulate the list of words. In particular, it realizes user requirements UR-12, UR-13, UR-14, UR-15.

Function This component manages the word list and the corresponding anagrams.

Subordinates It has no subordinates.

Dependencies There are no dependencies.

Interfaces The provided interface is offered as the following `public static` methods of the class *WordLibrary*:

- `int` *`getSize`*`()`, which returns the number of words in the list;
- *`String getScrambled`*`(int` *`idx`*`)`, where

Parameter *`idx`* with $0 \leq$ *`idx`* $<$ *`getSize`*`()` is the index of a word in the list

Returns random anagram of word at index *`idx`*.

- `boolean` *`isCorrect`*`(int` *`idx`*`;` *`String userGuess`*`)`, where

  Parameter *`idx`* with $0 \leq$ *`idx`* $<$ *`getSize`*`()` is the index of the correct word in the list

  Parameter *`userGuess`* is the word guessed by the user

  Returns whether *`userGuess`* is correct for index *`idx`*.

Typical usage of this interface is as follows:

1. Obtain the number of words in the list by calling *`getSize`*`()` once and storing the result;
2. Each round starts with calling *`getScrambled`* with a random index in the word list;
3. Repeatedly check user guesses for correctness by calling *`isCorrect`*.

Resources It requires no external resources.

For future versions, storing the word list in an external file is an option.

Processing This component does no autonomous processing; it only responds to calls on its interface.

Data It stores the list of words.

References

[1] Michael Jackson. *Problem Frames: Analyzing and Structuring Software Development Problems*. Addison-Wesley, 2000.

[2] Benjamin L. Kovitz. *Practical Software Requirements: A Manual of Content and Style*. Manning Publications Company, 1999.

Written by Tom Verhoeff (SET) as an example for a software engineering project at Eindhoven University of Technology, based on the *AnagramGame* sample Java program provided with the NetBeans IDE.