CS 6362 Software Architectural Design

SUMMER 2013

KWIC SOFTWARE SYSTEM

PHASE I - Final

ARCHITECTURE & IMPLEMENTATION

**Submitted To**

**Dr. LAWRENCE CHUNG**

**Associate Professor,**

**Department of Computer Science – Software Engineering,**

**The University of Texas at Dallas,**

**Richardson, TX-75080**

TEAM MEMBERS

Keerthikumar Ravichandran

Jayabharathi Sundararaj

Swapnil Mirajkar

Team Website

https://utdallas.edu/~kxr123330

ACKNOWLEDGEMENT

By signing my name below, I hereby agree and possess no dispute regarding my contribution to the project, through my attendance in team meetings, and the roles and responsibilities that I fulfilled during the current phase.

|  |  |  |
| --- | --- | --- |
| Name | NetID | Signature |
| Keerthikumar Ravichandran | kxr123330 |  |
| Jayabharathi Sundararaj | jxs127830 |  |
| Swapnil Mirajkar | sxm122731 |  |

Table of contents:

1. Introduction................................................................................................................................5

1.1 Purpose.................................................................................................................................5

1.2 Scope.....................................................................................................................................5

1.3 Definitions, Acronyms and Abbreviations.............................................................................5

2. Requirements Specification........................................................................................................5

2.1 Functional Requirements......................................................................................................5

2.2 Non Functional Requirements..............................................................................................6

2.2.1 Understandable............................................................................................................6

2.2.2 Portable........................................................................................................................6

2.2.3 User Friendly.................................................................................................................6

2.2.4 Performance.................................................................................................................6

2.2.5 Reusable.......................................................................................................................6

2.2.6 Responsiveness.............................................................................................................6

2.2.7 Adaptable......................................................................................................................6

3. System Description.....................................................................................................................6

3.1 Operating Environment........................................................................................................6

3.2 Functionality.........................................................................................................................7

4. Software Architecture Design.....................................................................................................7

4.1 Shared Data Design..............................................................................................................7

4.2 Abstract Data Design............................................................................................................9

4.3 Pipes and Filters Design......................................................................................................12

4.4 Implicit Invocation Design..................................................................................................13

5 Test Plan....................................................................................................................................14

5.1 Test Plan………………………………………………………………………………………………………………………14

5.2 Requirement Traceability Matrix………………………………………………………….………………………16

1. Introduction

The KWIC system is developed for easy the easy search of specific words, lines of characters, or other online entries. The KWIC (Key Word in Context) index system shall accept an ordered set of lines, where each line is an ordered set of words, and each word is an ordered set of characters. Any line shall be **“Circularly Shifted"** by repeatedly removing the first word and appending it at the end of the line. The KWIC index system shall output a listing of all circular shifts of all lines in ascending alphabetical order.

This document explains the life cycle, architectural design, functional requirements and non-functional requirements. The style, components, constraints, connections and design patterns are analyzed and an object oriented design in made.

1.1 Purpose

This document portrays the functional and nonfunctional requirements in detail. It also describes the architectural design of the KWIC software system and the implementation of it to achieve the goal.

1.2 Scope

The KWIC system serves as a base for the Search Engine. The document provides the architecture specification and various constraints. The various functionalities of the requirements are understood and the system is developed

1.3 Definitions, Acronyms and Abbreviations

Circular Shift: The first word of the line is removed and is appended at the end of the line including the delimiters.

2. Requirements Specification

2.1 Functional Requirements

FR1: Accept ordered set of lines, keystroke and pasted characters.

FR2: Accept inputs with maximum of 2048 characters ending with delimiters.

FR3: Output is circularly shifted index and are alphabetically ordered.

2.2 Non Functional Requirements

2.2.1 Understandable

The system must be in such a way that it is understandable by developers and users. The codes written for the system much be in a way such that it is readable, so that it can be reused.

2.2.2 Portable

The software system must be environment independent. It must be able to run while moving from one environment to another. The system must be compatible with all kind of operating system.

2.2.3 User Friendly

The system must be simple and the user interface must be intractable and must not be complicated.

2.2.4 Performance

The performance is the essential part of a software system. The system must be efficient in time and space.

2.2.5 Reusable

The system must be in such a way that it can be reused in future. Additional functionalities can be added to it and enhance its properties.

2.2.6 Responsiveness

The software system must be highly responsive so that it provides the output as quick as possible

2.2.7 Adaptable

The system must be adaptable and must be able to run in all kinds of environment and operating systems.

3. System Description

3.1 Operating Environment

The system does not require any specific operating environment. It runs in all kinds of operating systems such as Windows, UNIX, etc.

3.2 Functionality

The functionality of the KWIC system is as follows:

* Accepts inputs such as lines of characters, keywords, etc from the user.
* Circularly shifts the input by removing the first word and appending it to the end of the line.
* Alphabetically order the lines in ascending order.
* Finally display the output the result in the screen.

4 Software Architecture Design

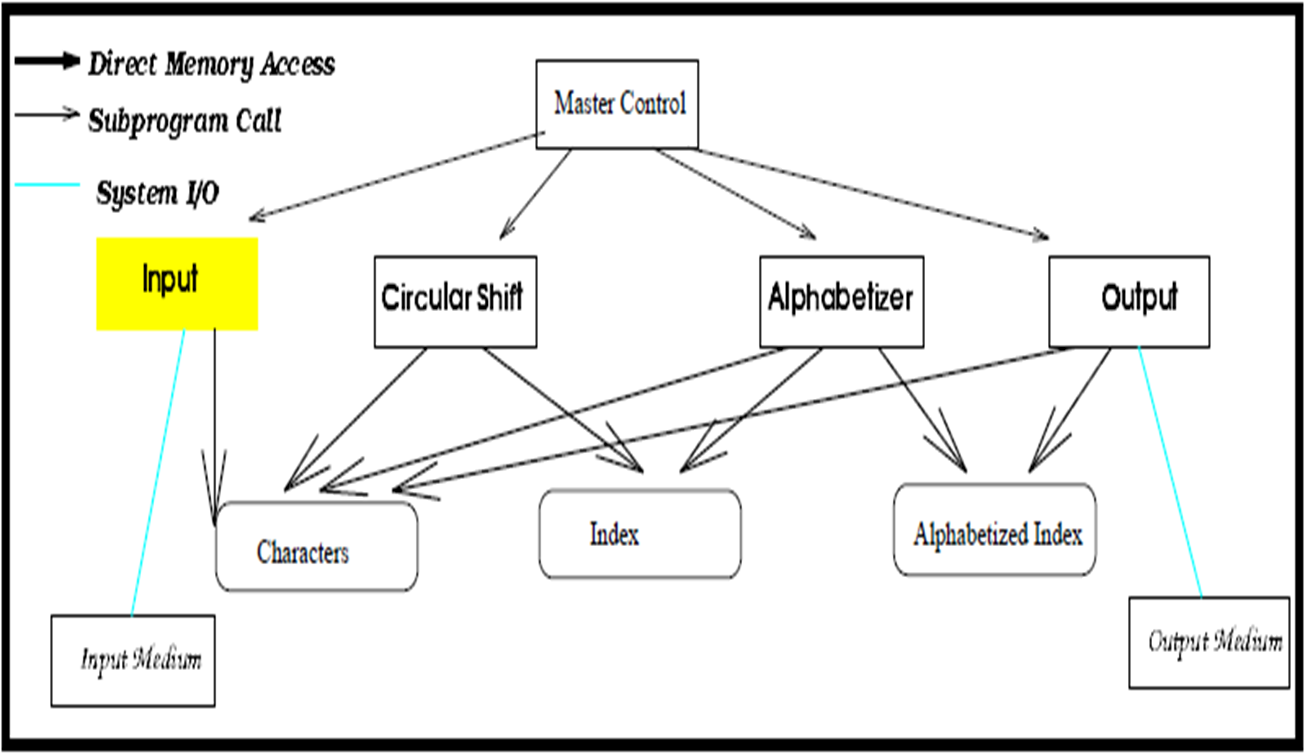
There are many different architectural styles available for a system. Some of the styles are Shared data design, Abstract data design, Implicit Invocation and Pipes and filters

4.1 Shared Data Design

4.1.1 Introduction

Data is communicated between the components through shared storage. Interaction between computational components and shared data is an unconfined read write protocol.

4.1.2 Architecture Design



4.1.3 Components

Input

Read Reads data from the input medium

Store Stores data lines as packets in main memory

Circular Shifter

readChar Read packet characters from “characters”

Prepare Record into “index” the starting index of the lines and offset for each word from the starting position, and conduct a virtual circular shift.

Alphabetizer

readChar Read packet characters from “Characters”

readIndex Read recorded Index from “Index”

alphabetize Converts Index to an alphabetized to by listing the circular shifts and then the result is stored in”Alphabetized Index”

Output

readChar Read packet characters from “Characters”

readIndex Read index from “Alphabetized Index”

Master Control

Batch mode, control the sequencing among four modules.

4.1.4 Connections

Direct Memory Access(DMA):

DMA helps in managing the time and memory efficiently because it avoids the copying of data every time a module is being used separately.

System I/O:

Input and Output comes from outside medium.

Procedure Call:

By obtaining an instance of that module and invoke the corresponding method/procedure.

4.1.5 Constraints

* Through shared storage, data is communicated between the components.
* To data access synchronization, special efforts need to be dedicated.

4.1.6 Advantages:

* Performance- As we use data sharing and not data multiplication this model offers the efficiency.

4.1.7 Disadvantages:

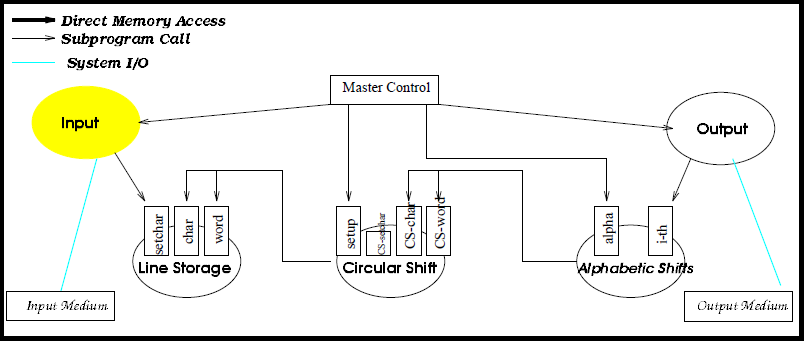
* Reusability is less - Since the components in this design are highly interdependent so it is difficult to reuse the components.
* Enhanceability is difficulty - Whenever new component has to be added in the system, it comes with a new data structure so the redesigning of the system is essential. It takes lots of time and difficulty to enhance the system.
* Modifiability- In this model, one component is strongly dependent of the output of other components. If the storage data structure is modified, it affects atleast one components

4.2 Abstract Data Design

4.2.1 Introduction

In ADT architecture design, the system can be viewed as a collection of ADT objects. Each ADT includes data objects, operations on data objects and properties of operations algebraic equations. Each ADT object also provides interfaces for other objects in the system.

4.2.2 Architecture Design:



4.2.3 Architectural Style:

Shared data design is used in the situation where components in the main

program with subroutines type of documentation. Decisions about data representations are a mutual property of the components that use the data.

4.2.4 Components:

Input

read Read Input string lines from the Input medium.

store Store data lines.

Line Storage

This method will create a vector to store the original output line.

* All the characters in the input lines will be converted into lower case, otherwise they remain the same.
* The word ‘a’, ‘an’, ‘and’, ‘the’,’of’ and ‘or’ will be extracted from the original string line, if no noise is allowed.

Circular Shift

This creates a circular shift for multiple string lines.

Alphabetic shifts

This creates alphabetized lines of the circular shifts and stores the result in vector. Strings are compared lexicographically and each character in the strings the character sequence represented by this string object is compared lexicographically to the character sequence represented by the argument string.

Output

This module gets a vector, which contains the output lines in alphabetical order and prints them in the output textarea line by line.

Master Control

This module makes procedure calls to all other modules.

4.2.5 Connections

* Procedure call: Through acquiring an instance of one module and invokes the corresponding methods/procedures.
* System I/O: This is input/output from outside medium.

4.2.6 Constraint

* Other components can only access data by invoking the corresponding interfaces, because of the information hiding of abstract data type.

4.2.7 Advantages

* Reusability: Components rely on external events and ar loosely coupled and can be reused by other systems easily.
* Enhanceability: We can include new components easily without affecting any existing components.
* Modifiability: The model is quite modifiable. Because the components are loosely coupled and data is kept in the control.

4.2.8 Disadvantages

* Performance: Due to reconstruction, this is slower than shared data design.

4.3 Pipes and Filters Design

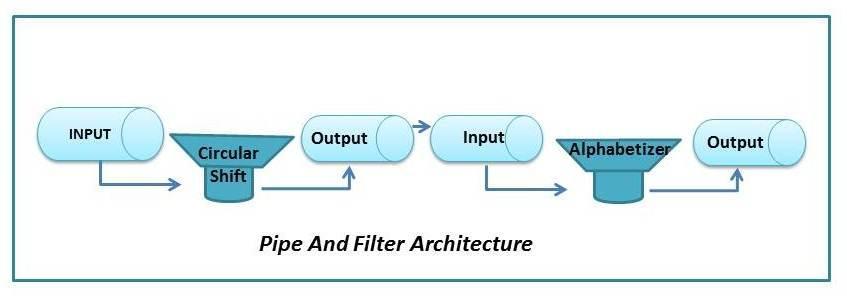
4.3.1 Introduction

A very powerful and robust architecture consists of any number of components that transform or filter data, before passing it on via connectors to other components.

Filters transform the received data and it can have any number of input pipes and any number of output pipes.

Pipe is the connector that passes data from one filter to the next. It is a directional stream of data that is usually implemented by a data buffer to store all data, until the next filter has time to process it.

4.3.2 Architecture Design



4.3.3 Architectural style:

This model architects the system into a set of sequential components. Each component does some transformation to the input stream of data and produce output stream of data.

4.3.4 Components

Filter Input Reads Input from Input media, processes and stores the data in a return pipe.

Filter Circular Shift Read input lines from an input pipe, creates and outputs circular shifts.

Filter Alphabetizer Reads circular shifts, creates and outputs.

Filter Output Read alphabetized lines and writes data to the output medium.

4.3.5 Connections

Pipes: Components can communicate with each other and sharing data through pipes.

System I/O: This is input from outside medium or Output to the outside medium.

4.3.6 Constraint

* Each filter can run whenever data to compute on has.
* Control is distributed.
* Process does not know the entity of their upstream and upstream processes.

4.3.7 Advantages

* Modifiable - Modifying a component does not affect other components, as long as there is no change in output. The internal data representation is not known to other components; therefore changing the implementation of a component does not affect any other component.
* Enhanceability - We can add or remove components easily.
* Reusability- Each component is distinct part of the system, So the components can be used by other system.

4.3.8 Disadvantages

* Enhanceability- It is difficult to implement an interactive system using pipes and filters.
* Performance- This uses much more space than other styles, because each component will replicate the data from its input to its output.

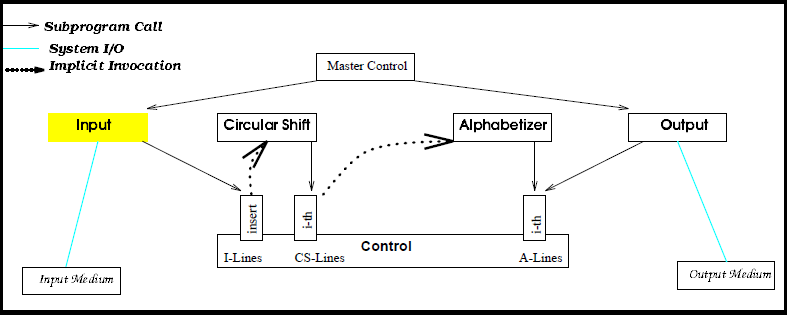
4.4 Implicit Invocation Design

4.4.1 Introduction

Implicit invocation uses a form component integration based on shared data. There are two important differences. First, the interface to the data is abstract.

Second, computations are invoked implicitly as data is modified.

4.4.2 Architecture Design



4.4.3 Architectural style

Every individual component has the ability to announce or broadcast an event.

* By associating a procedure with the event, other components in the system can register an interest in an event.
* The system will be quick enough to invoke all the procedures associated with the event as soon as the event is announced. Thus, an event announcement “implicitly”causes the invocation of procedures in other components.

4.4.4 Components

Input

Reads data lines from input medium through operation insert, a new line to the line,”I- lines” and then announces the event of completion of input.

Circular shifter

This module registers an interest in the event of completion of input by associating procedure I-th with it. When completion of input is announced, the control routine invokes procedure I-th.

Alphabetical Shift

This module registers an interest in the event of completion of noise elimination by associating procedure I-th with it. When completion of noise elimination is announced, the control routine invokes procedure I-th.

Module Output

This reads the alphabetized shifted lines from line buffer, A-lines, created by Alphabetical shift and displays the result to the output medium.

Master Control

This module explicitly invokes input and output through procedure calls.

4.4.5 Constraint

Computations are invoked implicitly as events are announced.

4.4.6 Advantages

Reusability- Components are rely on external events and loosely coupled and can be reused easily.

Enhanceability- Inclusion of new components is possible without affecting any existing components.

Modifiability- The components are loosely coupled and data is kept in control so the model is quite modifiable.

4.4.7 Disadvantages

Performance: It must wait for the events to be triggered so it works slower and tends to use more space than shared data model and the ADT model.

5. Conclusion

The KWIC index software system was implemented using JavaScript as per the architectural design. The functional and non-functional requirements have been fulfilled. Additional features are to be added in the upcoming phase.

Test Plan

**5.1 Test Plan**

A test plan is a document detailing a systematic approach to testing a system such as a machine or software. The plan typically contains a detailed understanding of what the eventual workflow will be. To ensure the quality of the given application, it should be thoroughly tested and verified. Proper testing requires a proper test plan. Test planning involves preparation of test strategies from the beginning of SDLC. During test planning, we focus on functional as well as non-functional aspects of testing.

* **Non-Functional Testing**: Non-functional testing is the testing of a software application for its non-functional requirements. NFR test cases will be prepared according to following NFR criteria:
* Understandability
* Portability
* Good Performance
* User-friendliness
* Responsiveness
* Adaptability

|  |  |
| --- | --- |
| Test Case Identifier: | NFR7\_Performance |
| Test Items: | Output Area |
| Input Specifications: | Input text with lines = 10 |
| Output Specifications: | An input of 10 lines was tested and output was obtained in less than a second |
| Environmental Needs: | Web browser on client PC or Laptop |
| Special Requirements: | None |
| Inter-case Dependencies: | This test case relies on the availability of KWIC system |

* **Functional Testing**: Functional testing is the testing of a software application for its functional requirements. FR test cases will be prepared according to following FR criteria:
* Input Verification
* Circular shifting each line
* Delimiter
* Input methods
* Max Characters
* Output Verification

|  |  |
| --- | --- |
| Test Case Identifier: | FR7\_Max characters |
| Test Items: | Input Area |
| Input Specifications: | Enter more than 2048 characters (including the delimiters) |
| Output Specifications: | User should NOT be able to input more than 2048 characters |
| Environmental Needs: | Web browser on client PC or Laptop |
| Special Requirements: | None |
| Inter-case Dependencies: | None |

## Requirement Traceability Matrix

**5.2 Requirement Traceability Matrix**

A traceability matrix is a document, usually in the form of a table that correlates any two baseline documents that require a many-to-many relationship to determine the completeness of the relationship. It is often used with high-level requirements and detailed requirements of the product to the matching parts of high-level design, detailed design, test plan, and test cases.

A sample traceability matrix is as shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Cases** | Test\_1 | Test\_2 | Test\_3 |
| **Requirement Number** |  |  |  |
| 1.1 | X |  |  |
| 1.2 |  | X |  |
| 1.3 |  |  | X |

A requirements traceability matrix may be used to check to see if the current project requirements are being met and to help in the creation of a request for proposal, software requirements specification, various deliverable documents, and project plan tasks.