

Software Architecture and Design

Project 2

Infominer Software Architecture for Web-based Search Engine

Document 1 Architectural Specification

**CS 6362 - Software Architecture
Dr. Lawrence Chung**

Team Tripod

Avinash Chiganmi
Phaneesh Kuppahalli Lakshminarayana
Sameer Agrawal

Table of Contents

1. INTRODUCTION	3
3. ARCHITECTURAL SPECIFICATION.....	4
4. DESIGN DIAGRAM.....	8
5. CONCLUSION.....	12
REFERENCES.....	13

Table of Figures

Figure 1. Index Component	<u>8</u>
Figure 2 Search Component	<u>9</u>
Figure 3 Index Component Class Diagram.....	<u>11</u>
Figure 4 Search Component Class Diagram.....	<u>12</u>

1. INTRODUCTION

The objective of this project is to design and build the Infominer software system, which is a growing web search engine, making use of the KWIC software system, built in the first part of the project.

This document describes the requirement specifications and the architectural specification of the Infominer software system. This includes the style, components, connections, constraints and the rationale for using the particular style.

2. REQUIREMENT SPECIFICATION

2.1 Functional requirements:

- Info-miner shall accept a list of keywords and return a list of URL's whose descriptions contain any of the given keywords.
- Info-miner shall use another software system as a component, KWIC, in order to efficiently maintain a database of URL's and the corresponding descriptions.
- KWIC shall accept an ordered set of lines, where each line consists of two parts:

- the URL part, whose syntax is

URL ::= 'http://' identifier . ' identifier . ' ['edu' | 'com' | 'org' | 'net']

Identifier ::= { letter | digit } +

letter ::= ['a' | 'b' | ... | 'y' | 'z' | 'A' | 'B' | ... | 'Y' | 'Z']

digit ::= ['1' | '2' | ... | '9' | '0']

- The syntax of the descriptor is as follows
descriptor ::= Identifier { " Identifier } *
- The descriptor part of any line shall be "circulated shifted" by repeatedly removing the first word and appending it at the end of the line. The KWIC index system shall output a list of all circular shifts of the descriptor parts of all lines in alphabetically ascending order, together with their corresponding URLs. No line in the output list shall start with any noise word such as "a", "the", and "of".

- KWIC shall allow for two modes of operation:
 - for building an initial KWIC indices;
 - for growing the indices with later indices
- Case sensitive search: The system shall store the input as given and retrieve the input also as such

2.2 Non-functional requirements:

- **Easily understandable** – the system should be easy to learn and understand
- **Portable** – the system should run on many platforms, browsers and operating Systems
- **Enhanceable** – the system should allow for enhancement without major code rewrites or architectural changes
- **Reusable** – the components of the system should be reusable
- **Good performance** – the system should provide good performance on all Platforms
- **User-friendly** – the system should be intuitive and easy to use
- **Responsive** – the system should respond to user actions quickly
- **Adaptive** – the system should be able to adapt to changes

3. ARCHITECTURAL SPECIFICATION

The architectural specification used is Data Abstraction and Object-Oriented Organization. This architecture depends upon the concepts of Data Abstraction, Data Encapsulation and Object Oriented Design to decompose the solution into components called *Objects*. Data and operations are encapsulated into an abstract data type. The components of this style are the objects and connectors operate through methods invocation. Objects are responsible for maintaining the integrity of a resource and the representation of the object is hidden from other objects – Data Abstraction and Encapsulation.

- **Style :**

Each module provides interfaces through which other components can access data. The data is hidden in each component. So we can organize several set of objects and make each object can handle the representation of the data, and provide procedural interface for accessing the data.

- **Components :**

- **Module Input**

This module accepts the input from the user.

§ Procedure *readInput* : Reads input string lines from the input text box.

§ Procedure *tokenizeTextURLToLines()* : Validates the input lines by calling “*isValid()*” of the Validator module and stores data lines by calling the “*add*” of “*lineStorage()*” module.

- **Module Validator**

§ Procedure *isValid()*: calls “*isValidURL()*”, “*isValidProtocol()*”, “*isValidTerminator()*”, “*isValidIdentifier()*” which are its own methods and validates the input.

- **Module LineStorage**

This module stores the input lines into an ArrayList.

§ Procedure *add (lineURL line)*: Add each input line into an ArrayList.

§ Procedure *getLineURL (int i)*: Returns the i^{th} entry in the ArrayList.

§ Procedure *getLines()*: Returns the lines.

§ Procedure *setlines(ArrayList list)*: Sets the lines to list.

- **Module CircularShift**

This module creates a circular shift for multiple string lines.

Circular Shift means repeatedly removing the first word and appending it at the end of the line until a circular is reached.

§ Procedure *circularShift (LineStorage lineStorage)*: Constructs the circular shift and put the results lines into a new ArrayList.

§ Procedure *getCircularShiftedLines ()*: Returns the ArrayList which contains circular shift lines.

- **Module NoiseEliminator**

This module removes the noise words.

- § Procedure *getNoiseEliminatedList()*: returns the ArrayList, eliminating the lines beginning with noise words.
- § Procedure *eliminateNoiseLines(CircularShift circularShift)*: eliminates the lines beginning with noise words.

- **Module AlphabeticShifts**

This module creates alphabetized lines of the circular shifts and save the result in an ArrayList.

- § Procedure *sort (NoiseEliminator noiseEliminator)*: Constructs the alphabetized shifts and puts the results lines into a new ArrayList.
- § Procedure *getSortedList ()*: Returns the ArrayList which contains alphabetized lines.

- **Module LineURL**

This module consists of the description and the URL.

- § Procedure *getLine()*: returns the description.
- § Procedure *getURL()*: returns the URL.
- § Procedure *getId()*: returns the ID.
- § Procedure *setLine(String l)*: sets the description to String l.
- § Procedure *setURL(String url)*: sets the URL to String url.
- § Procedure *getSetId(int id)*: sets the ID to id.
- § Procedure *compare(Object o1, Object o2)*: returns true if the description in o1 is same as description in o2.

- **Module Output**

This module prints the alphabetized circular shifted lines in the output textbox line by line.

- **Module CloudscapeDB**

This module establishes the connection with Cloudscape database for storing and retrieving the URL's and their descriptions.

- **Module IndexBean**

This module is the MasterControl for Indexing the input URL's and descriptors and is responsible for making procedure calls to all the

other modules. This module is also responsible for removing out of date URL's.

- **Module Expression**

This module is responsible for transforming the input search string into the corresponding SQL query.

§ Procedure *handleInputExpression()*: converts the input expression into the corresponding SQL query by calling “*getQuery()*”, “*removeDoubleNegation()*”, “*getChangedExpression()*”, which are its own methods.

- **Module SearchBean**

This module is the MasterControl for searching the database based on the input keywords and displaying the corresponding URL's and descriptors. It is responsible for making procedure calls to all the other modules.

- **Connectors :**

- **Procedure Calls**

The procedure call is made through acquiring an instance of one module and invokes the corresponding methods/procedures.

- **System I/O**

This is Input/Output from the external entity which is the user.

- **Constraints :**

Because of the information hiding of objects, other components can only access data by invoking the corresponding interfaces.

4. DESIGN DIAGRAM

- INDEX COMPONENT

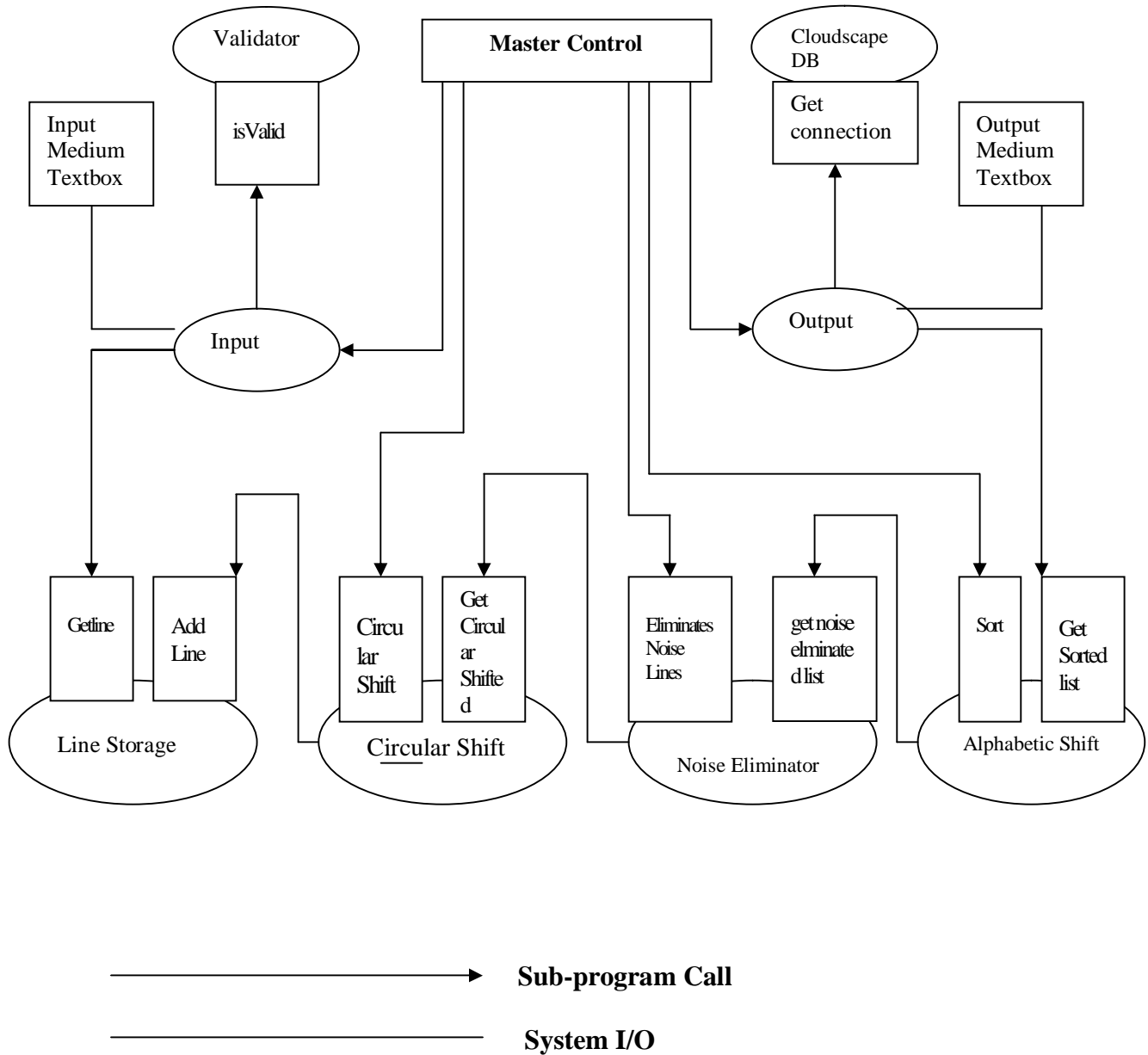


Figure 1. Index Component

- **SEARCH COMPONENT**

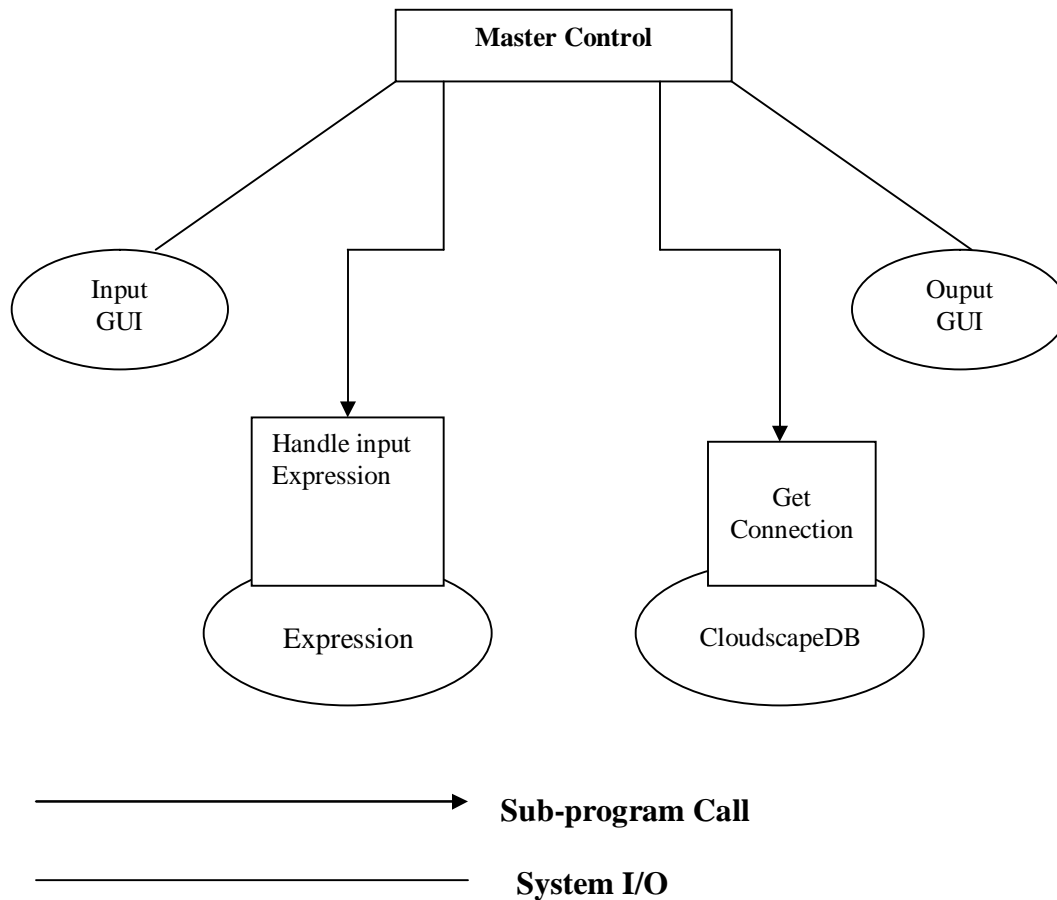


Figure 2. Search Component

The advantages and disadvantages of using the Data Abstraction and Object-Oriented Organization architectural style are as described below:

Advantages:

- Data encapsulation : It hides the implementation details which allow the object to be changed without affecting its clients.
- Reuse: It provides a higher degree of reusability.
- Adaptability: It is more adaptable to changes.

Disadvantages:

- Space Requirement is higher as compared to other models such as Shared Data Model..
- Response time is higher as compared to the Shared Data Model because of reconstruction.

Rationale for the design decision:

Based on the functional and non-functional requirements, the Data Abstraction and Object-Oriented Organization architectural style is best suited for the Infominer software system as described below:

- The model has higher portability as compared to other design models.
- Incremental development is possible – hence enhanceability is supported.
- Reusability is higher as every module encapsulates its own data and is loosely coupled with other modules.
- User friendly interface can be easily provided.
- Since the data is hidden in each module, modification to any module in either algorithmic or data representation will not affect other modules. Thus the model is modifiable and adaptive.
- Though response time and space requirements are higher, since this is a small system, the advantages outweigh the disadvantages.
- Concurrency is provided by using **J2EE**.
- Client Server architecture is supported using **J2EE**.
- Front end for the project is provided using **JSP/HTML**.
- Database used is **Cloudscape**.
- The middle tier is the **EJB** in the SUN **J2EE** server.

Class Diagrams

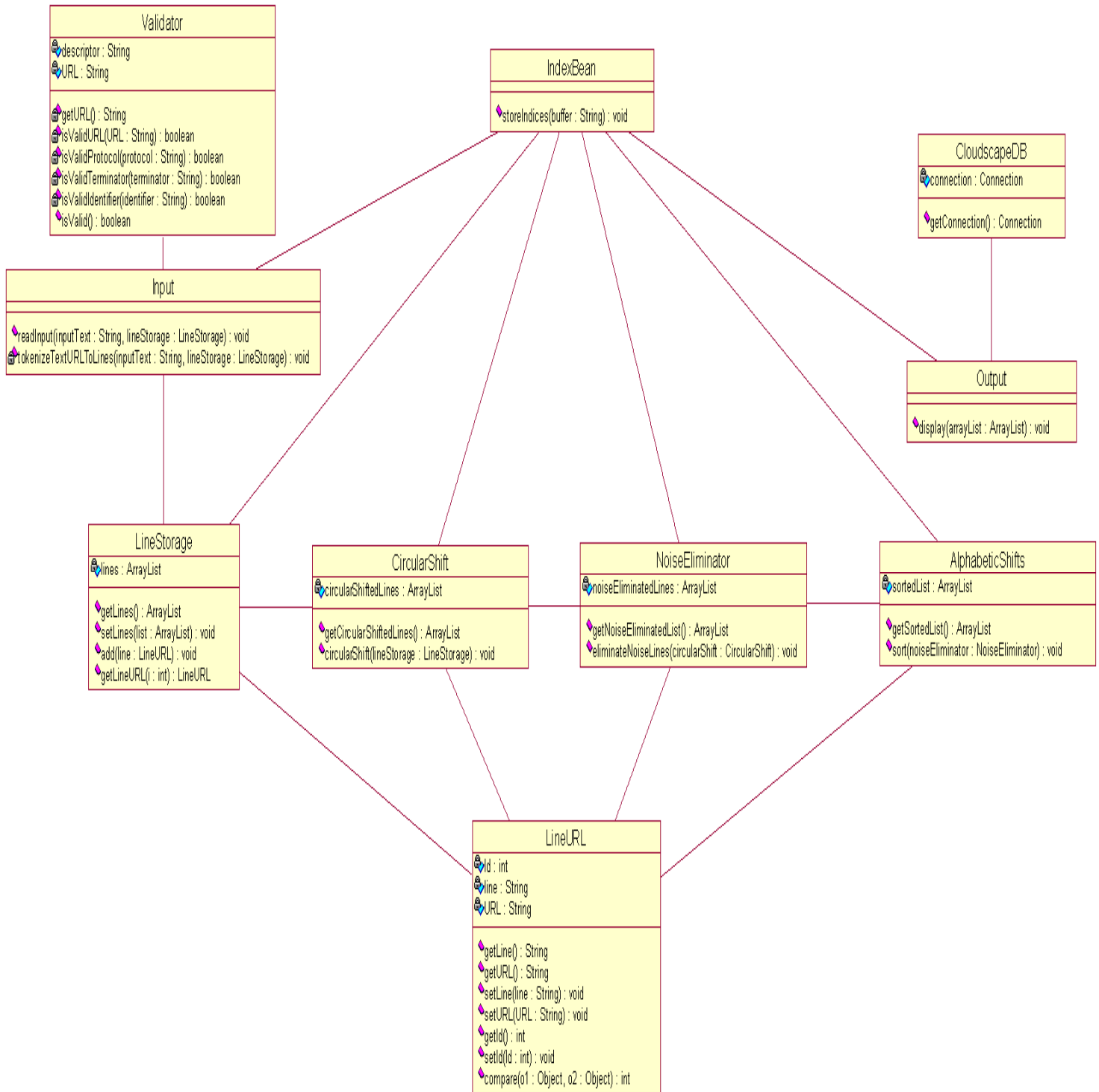


Figure 3. Index Component Class Diagram

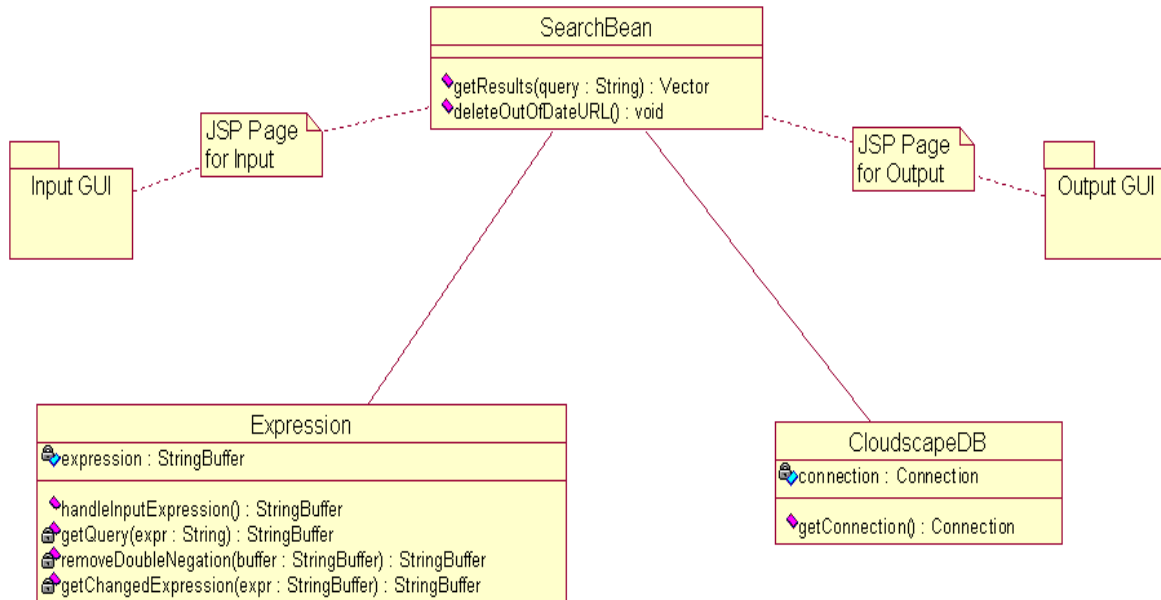


Figure 4. Search Component Class Diagram

5. CONCLUSION

This document describes the architectural specification of the project. It also describes the style including the components, components, connections and constraints. It also explains how the functional and non-functional requirements are satisfied by the model. The rationale and traceability between requirements and architecture specification are also established.

REFERENCES

- [1] Shaw, M, Garlan, D, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.
- [2] University of Texas at Dallas, “System/Software Architecture”, January 5, 2005, <http://utd.edu/~chung/SA/2.4md3.pdf>