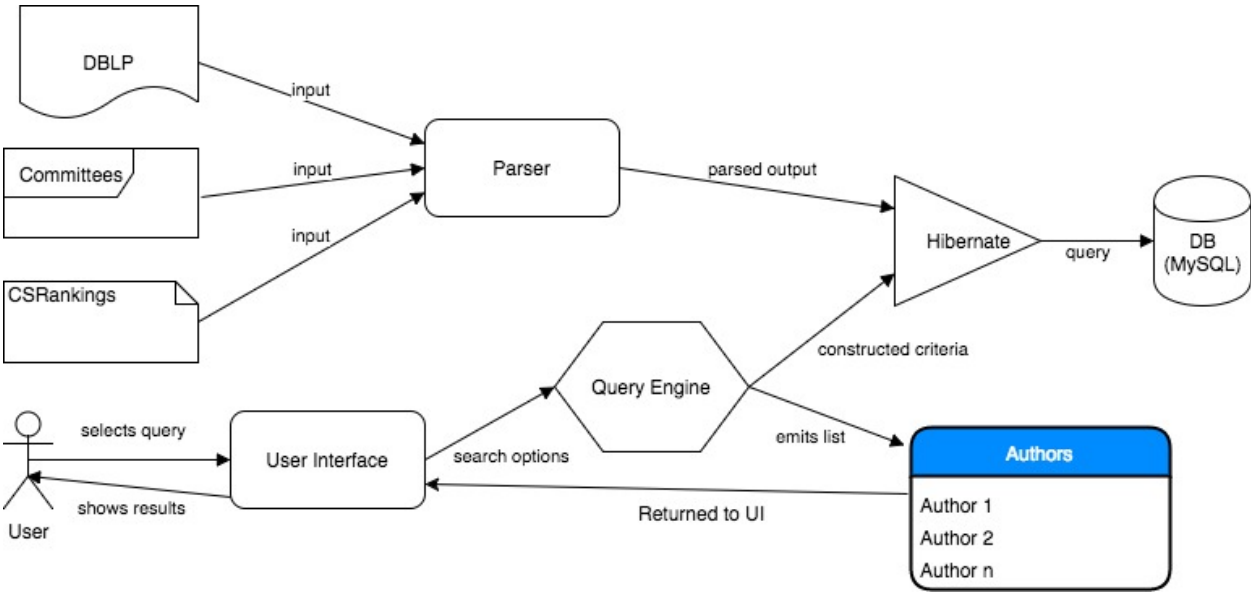
TEAM 6
Section 1

System Functionality:



As shown in the diagram, we have divided our system into three major components namely, the Backend, the Query Engine and the User Interface.

The *Backend* constitutes of a universal parser, that handles multiple types of input data sources and generates a consolidated schema in the database. We use hibernate for all the database related operations and it's easily configurable in the hibernate.cfg.xml. For now, the data has been filtered for the specified journals and conferences thereby, improving performance of the system as well as the response time for the queries.

The *Query Engine* deals with the various table entities and their data access objects. These data access objects are implemented to provide interface for storing and retrieving data. When a query is requested based on the user input, the query engine simply translates this into a set of criterions and calls the appropriate entities. Due to hibernate, the results are returned as a list of objects and can be easily displayed on a UI. We have also provided an advanced search option where multiple criterions can be selected to create a customized query by the user himself in real time. This provides flexibility to the system as well as ease of access.

The *User Interface* provides the visual entry point for a user to our system. The user needs to login using authorized credentials and only then can access the system. Once logged in, the user can simply select any of the 4 queries which have been pre built or make his own using the advanced search option. The author details are returned as a table on the screen and can be used by chairs for committee selection. User can also view the details like Affiliation, Country, Department of any author.

TEAM 6
Section 1

Enhancements & Changes:

As a requirement for phase 4, we applied the following enhancements to our system:

CSRankings Data:

In order to add more details about the authors, we used a compiled sheet of data with comma separated values to append details about them. We used our parser class to add the additional parsing of the CSRankings data. It simply constructs a table using hibernate for the additional information about the authors. This information is then used by our Query Engine which populates the additional details of the Author. On the User Interface side, we have introduced a new screen to view the details of a selected author. This screen displays data from CSRankings like Affiliation, Country, Department as well as DBLP like number of conference papers published, number of journal papers published and list of DBLP paper keys.

Better Error Handling:

As pointed out to us in HW4, we had a few loose ends that didn't handle exceptions or errors gracefully. We now have these handlers in place and will ensure a graceful exit of the system on any of the unforeseen errors.

User Interface Changes:

Some enhancement suggestions and bugs were pointed out to us in HW4 for the User Interface which we have fixed in this Phase. These changes include maintaining history for Back button on the search results page (i.e., if the search is invoked from Advanced Search tab, clicking on Back button should go to Advanced search tab and similarly for Quick Search tab), added dropdown with multiple select check boxes for Journal/Conference names on Advanced Search tab (to guide user for valid input and available conference/journal names), validated inputs for Author Like and Title Keywords field properly to handle exception scenarios (For example, entering '%' in Author Name or '*' in Title Keywords). We also changed the default spinner value for Consecutive Year constraint to 0 as suggested to us, made search results table uneditable. One of the enhancement suggested to us was addition of a way to view Author details. This was accomplished by addition of Author Details screen after using CSRankings data which had more information about Authors as compared to DBLP.

TEAM 6
Section 1

Data Changes:

In Phase 3, we had used entire DBLP data for our application which consisted of 10+ million Authors, 3+ million Papers from Conferences and Journals. The parsing time for entire DBLP database into MySQL via hibernate was around 15-20 minutes. Also, the application returned results in under 20 seconds even for complex queries. During HW4 when multiple users were accessing the system, there were slight performance issues in the application and some of the queries were timing out or the data was too much for hibernate to handle. Hence, we decided to filter out selected conferences and journal papers for our application. This improved the application performance drastically and we get the results back almost instantaneously. Also, the parsing has also become much faster and it takes under 2 minutes.

Test Cases:

We have added few additional test cases for the enhancements/bug fixes covered in Phase 4.

CheckStyle Update:

Previously we rely on Java compiler warnings to do lint and doclint. It works but we wanted better. In phase 4 we adopted Google Java Style, to achieve this, we had the following changes:

- Updated Jenkins CheckStyle job with a customized checker.
- Added customized Eclipse formatter to help us format code.
- Added customized Eclipse import organizing rule to help us organize imports.

The Checkstyle job reports warnings on code format, variable naming, Javadoc format etc, and we've eliminated 95% of the warnings.

Code Refactoring:

We had refactored most parts of the code in Phase 3 itself when we had some additional time. Some of the refactoring included converting the Java project to Maven project for easy installation and use, removed extra print statements and commented code, redundant variables, created inline local variables and methods, method and variable names.

Design Patterns:

We use the following patterns in our code:

Singleton Pattern:

Singleton pattern restricts the instantiation of our parser and ensures that only one instance of the class exists in the java virtual machine. The singleton parser class provides a global access point to get the instance of the class.

TEAM 6
Section 1

Package: com.neu.reviewerfiner.backend

The use of Hibernate enabled us to use the following design patterns in our system:

Factory Pattern:

We create the SessionFactory object without exposing the creation logic to the client and refer to newly created object using a common interface.

Package: com.neu.reviewerfiner.hibernate

Domain Model Pattern:

An object model of the tables that incorporates both behavior and data for all entities used in our system.

Package: com.neu.reviewerfiner.hibernate.dao, com.neu.reviewerfiner.hibernate.conf

Proxy pattern:

In hibernate, a class represents functionality of another class. It creates objects having an original object to interface its functionality to outer world.

Package: com.neu.reviewerfiner.backend, com.neu.reviewerfiner.hibernate

Github Issues:

As discussed previously, some of the issues filed as part of the Homework 4 were related to User Interface and some were related to performance because of huge data. We have fixed all the bugs reported to us as part of Phase 4. We also tried to incorporate as many enhancement suggestions as possible. There were two or three issues which we were unable to reproduce and we didn't get enough information back from the issue reporter to debug/fix it. A few other bugs were invalid or duplicate (same bugs reported by same person). Apart from this, there some clarification questions regarding the application like working of similar authors. We have addressed these issues in our User manual.