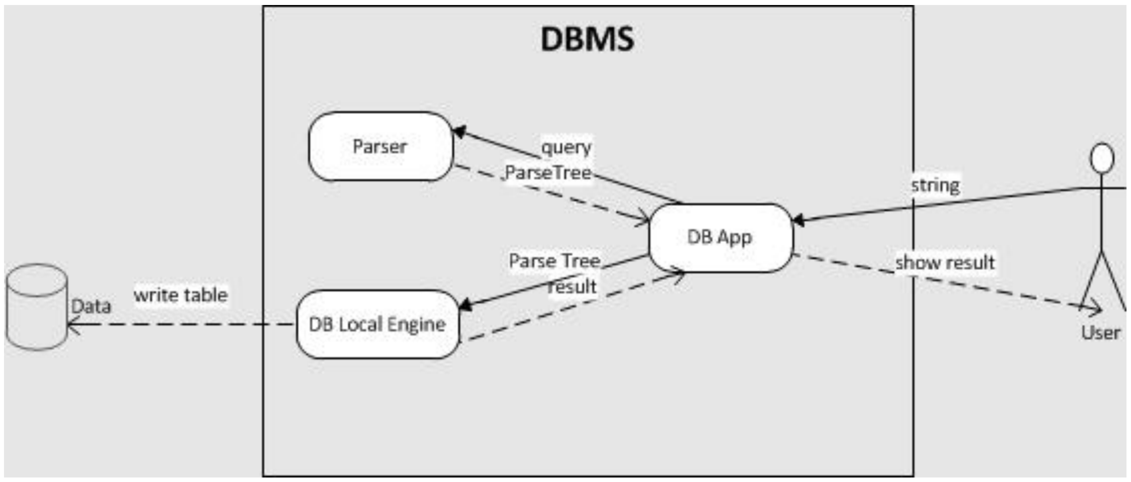
Section I

Our project aims to provide a database where the user can interact at the application level to primary register for classes, create new students, courses and review the courses taken for a given student or the students taking a given course. Our database will contain the information of the students, courses and the student registered in specific courses.

The system will help the user to easily get the details such as number of students in each/specific course, courses taken by each student or how many and which students are enrolled in each course, the courses under each/specific department. This kind of application will be most useful for students to look up for the courses that they are registered for, or register new courses, what courses are being offered by the department.

Through the course of this project we aim to learn how to build a database engine, create a parser and design an application that interacts with the parser and the database engine to give the desired output to the user.

Section 2

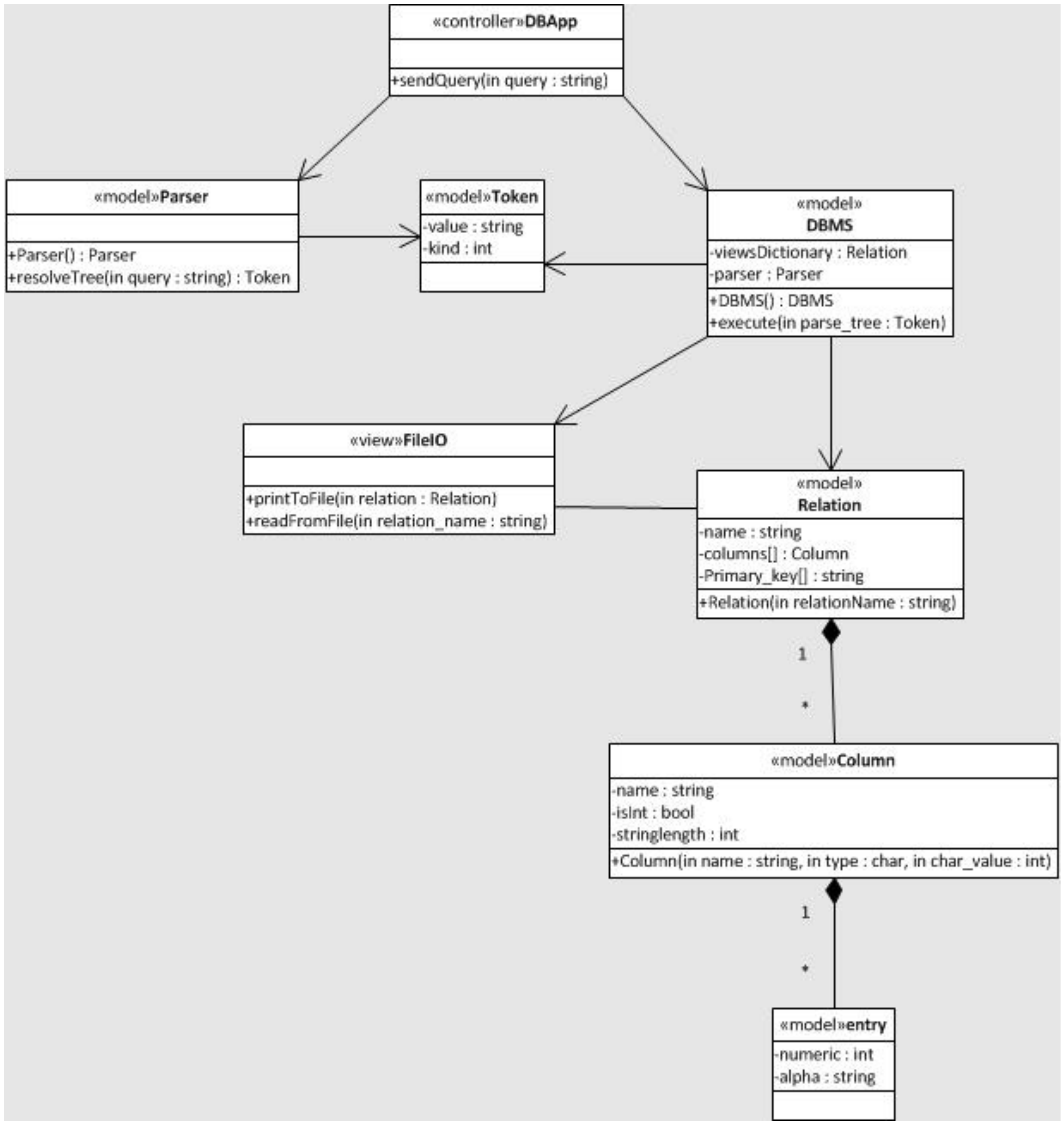


High level entities

This figure shows the entire flow of information inside the system. The user will introduce a query using a CLI interface, the DB App will translate this query into our pseudo SQL type query and pass it to the parser. The parser will generate the Parse tree using the grammar described in the Project Documentation and return it to the DB app. The DB APP will pass this tree to the local engine, which will resolve the query, update the table in memory and/or files (if needed) and send back the result to the DB app which will show the final result to the user in the CLI interface.

Section 3

Model



Domain Model

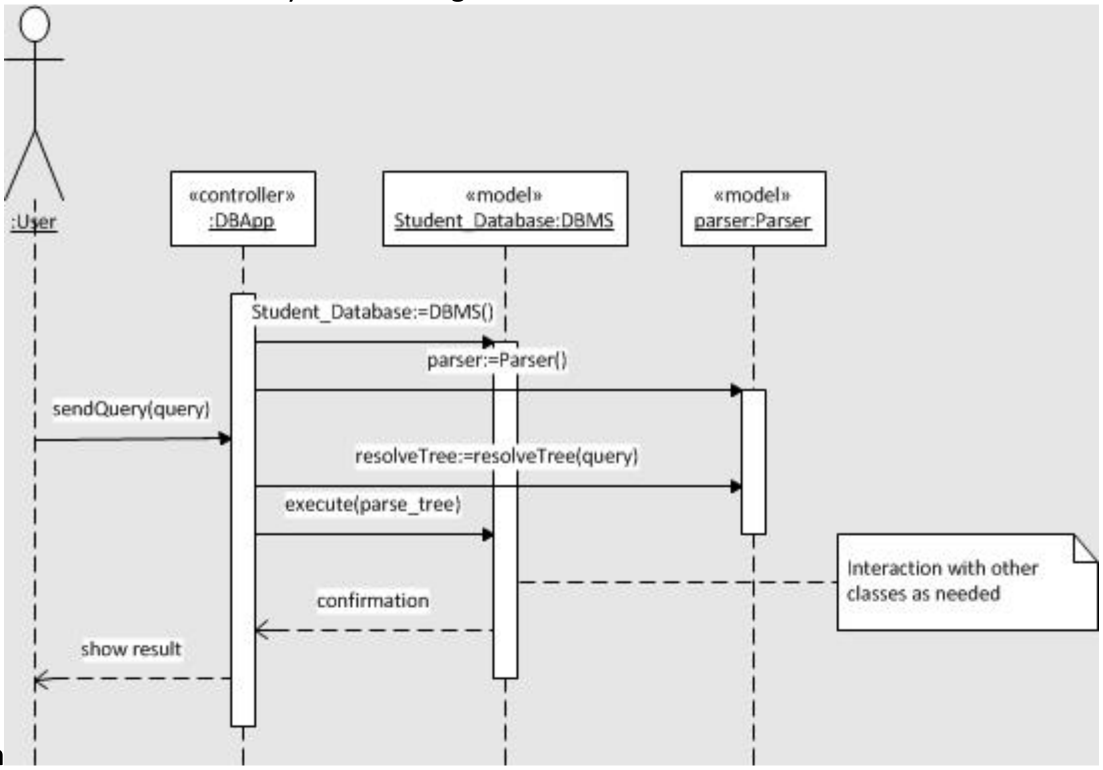
This figure describes in high level the functionality of the entities and the classes we are going to use in the project. Classes have only the most important attributes and functions we will use. The actual Parsing (class Parser) and query execution (class DBMS) are not totally detailed.

Usage/Initial Configuration*

DBApp: This is the starting point of the system. It will initialize a DBMS object, load initial data in the database and show the menu to the user.

DBMS: It will initialize a Parser object.
Relation: It will initialize a Relation object with the name passed as parameter.
Column: It will initialize a Column object with the parameters passed.
Row: It will initialize a Row with the values passed.
*Objects omitted do not need any initial configuration

Interaction



Sequence Diagram

This figures show the sequence of function calls in our different objects that a user input (query) will start.

Post Production Notes

We didn't made big changes to the original design. Most of the changes were refinements to the original design so we can ease the coding stage and improve performance and design of the whole system. We can list some of the more important:

- We add a class entry to store the actual data of the database in the memory. At first we thought in store this data only as string but we decided is better designed if we store them as an object where we will have the possibility to specify if what is stored there is an INTEGER or a VARCHAR type and we can actually store this two types of data as C++ ints or C++ strings. We think that was a good decision given that it help us keep track of the actual value of the data and is a best design than just store everything as string.

- We used columns as vector of entry objects. Therefore the object columns is the actual repository of the data. And a relation is a vector of columns. We decide to do not specify rows as a different class but just as a vector of entry instances. We think this decision help us do some operations in an easier way, while another ones were a little more difficult, but we believe that if we had decided to use rows as our principal repository of data we might have faced similar problems.
- We decide to use another class to interact with the File system. We think it is a better design decision.
- We decide the DB app should act as the controller of the program instead of the DBMS acting as it. It is the DB app will call the parser, receive a response and then call the DB engine and show results. At first we had the DB app calling directly the DBMS and this calling the parser, receiving the response and resolving the query. We think both approaches are valid, but given we first developed only the DBMS and then the Parser, this approach has the advantage that we could use it to test this parts separately, so that we could test and use the DBMS without having the Parser working and vice versa.