

# An Object-Oriented Framework for Predicting Student Competency Level in an Incoming Class

Suresh Kalathur

**Abstract**—When students of various backgrounds enroll in a class, it is hard for the instructor to infer the composition of the class and the background level of all the students. This paper discusses an object-oriented model coupled with data mining techniques that will predict the class configuration based on the course prerequisites and the prior courses taken by the students. Equipped with this information, the instructor can undertake remedial measures by supplementing the lecture material with the required topics which the students were lacking from their previous courses.

## I. INTRODUCTION

IN this paper, we present a framework that will aide an instructor in understanding the composition of their incoming class students. This framework is primarily geared towards graduate degree programs catering to the needs of continuing education students. Usually, the students, in these types of programs, are working professionals who take one or two classes each semester. Sometimes, depending on their work schedule (or personal reasons), they may return after a hiatus to continue with the program. Given this scenario, it is practically impossible for an instructor offering a course to assume that all the students in the class will be well versed with the prerequisite material required for their upcoming course. Consider the simplest scenario where there is only one prerequisite course required. The instructor is looking for the following answers: Did all the students take the prerequisite course? Did a majority of the students take the prerequisite with the same instructor and in the same semester? If that is the case, the instructor has a clear picture of what can be expected from the students. But this is hardly the typical scenario in our case. The students may have taken the prerequisite course with various instructors during different intervals. The course syllabi may have varied from one time to the other. The later versions might have covered recent developments in the field which were non-existent when the same course was offered during the earlier period.

Another interesting scenario to consider is the case when a majority of the students didn't take the prerequisite course. But they are interested in taking the instructor's course since they feel comfortable with the course syllabi. In this particular case, the instructor would like to know whether the students gained the core required knowledge from other

courses they have taken in the past? What are the common skills that the students enrolled in the class have? Are they enough to meet the prerequisite knowledge required? Having equipped with this information before hand, the instructor can take remedial measures as appropriate during the course to address and close the knowledge gap that would have existed otherwise.

In the following section, we will discuss the various entities and their attributes. The model can be viewed from a relational database perspective or from an object-oriented view. A *Course* object captures the static part of a course. Each *Semester*, there will be *Course Offerings* capturing the dynamic content of the *Course* that varies from one offering to another. The *Course Syllabus*, as usually described in the *Course Catalog*, is associated with its respective *Course*. If the syllabus for any offering differs from the one associated with the course as is normally the case, the new course syllabus will then be linked directly to its specific course offering.

## II. DATA OBJECT MODEL

In this section, we look into the data model from an object-oriented perspective and describe the various classes and the associations among them. Each class represents an entity and thus a table in the relational database. The associations between the classes are captured with the foreign key constraints in the database. Since Java is used in implementing the classes, the object-relational persistence is achieved by making use of the Hibernate [1] framework to maintain the consistency between the Java classes in our implementation and the tables in the relational database. The Hibernate schemas showing the mapping between the object model and the relational model are illustrated in Appendix A. The following are the core classes capturing the data model following the Unified Modeling approach [2].

### A. *Course Syllabus*

The syllabus for a course is usually a text document (ASCII, HTML, Word, or PDF). Once the course syllabus is established for a course or any course offering, data preprocessing is done to extract the relevant data from the syllabus. Text mining techniques are then used (word extraction, stemming, frequency counts) to compute the important concepts of this course [5]. All this information is then stored with the course syllabus and used whenever required.

### B. Courses, Semesters, and Course Offerings

Associated with each semester is a list of courses offered during that semester. This one-to-many relationship between the *Semester* class and the *CourseOffering* class is shown in the UML class diagram in Figure 1. Each course offering is for the respective course which is captured by the relationship between the *CourseOffering* and *Course* classes. Given a particular course, we can examine all the offerings of the course along with the semester in which it was offered.

### C. Students, Course Offerings, and Instructors

For a given student, we would like to examine all the courses they have taken so far. The many-to-many relationship between the *Student* class and the *CourseOffering* class is the key association in our model. We can provide the list of courses taken by any student during a particular semester or during the past ‘x’ semesters by navigating this association. Also, associated with each course offering is the *Instructor* who taught the course. Figure 1 captures all these relations.

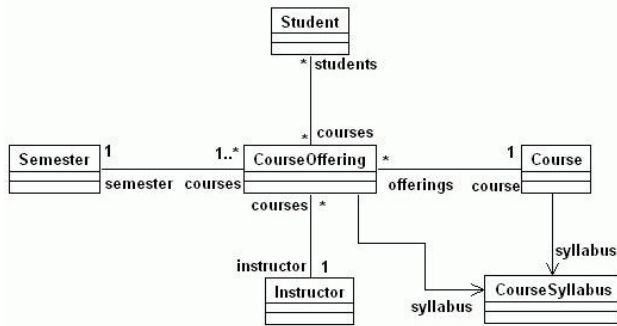


Fig. 1. Class Diagram for the Data Model

### III. BUILDING STUDENT PROFILE MODEL

The primary use case of the system is to build the student profile for a particular course offering and present the results to the instructor teaching the course.



Fig. 2. Use Case Diagram

The first step is to look at the prerequisites for this course, retrieve the course syllabi for those prerequisites, extract important concepts for these syllabi and build a unique document vector combining the required concepts. Since the

prerequisites are part of the course syllabus object, we first look at the course object of this offering, and ask its course syllabus for the required prerequisites. Since the information is stored in a text field (*prereqs*), the course numbers are parsed and the appropriate course objects representing the prerequisites are loaded and returned. For each of these courses, the document vector associated with the course content is either built or retrieved. The document vectors are then aggregated to build the profile of the required prior knowledge for this course offering. The interaction between the various objects is shown in Figure 3.

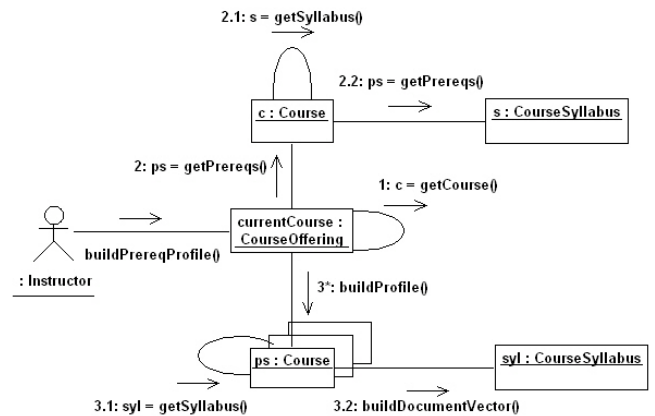


Fig. 3. Collaboration Diagram – Building Prerequisites Profile

The next step is to iterate over the students enrolled in this course. If all the students took all the prerequisites within the past year or two, then there is a high likelihood that all the students were well versed with the required prior knowledge for this course. This scenario would occur when the course syllabi of the various offerings of this particular course are almost similar to the prescribed course syllabus itself. This is particularly true with courses which have more or less a well established syllabus (for example, Compilers, Operating Systems, etc.) However, for courses which rely on state of the art topics, there will be substantial variations in the syllabus from one offering to the other. In this case, we would like an estimate about the students who took the prerequisites but are outdated. The above two scenarios can be handled in a uniform manner by first computing the document vector from the course syllabi of each student’s course offerings of the prerequisite courses. Then the similarity index is computed by comparing this document vector with that of the prerequisites.

The other scenario to consider is the case when some of the students didn’t take some or all of the prerequisite courses. For such students, the document vectors of all the courses they took are computed and compared with the prerequisites document vector to find the similarity index. At the end of this process, we have the document vector for each student and the similarity index relative to the prerequisites. The higher the similarity index, the more appropriate is this course offering for the enrolled students.

Figure 4 shows some of the interactions between the various objects while building the students' profiles.

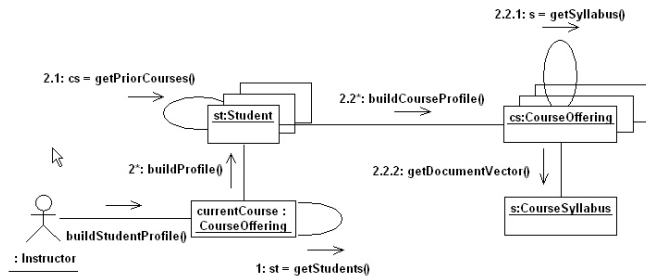


Fig. 4. Collaboration Diagram – Building Students Profile

#### IV. MODEL FEEDBACK

Now that the students' profiles have been determined, the model should provide the feedback to the instructor. One such feedback is a sorted listing of the similarity index for each student enrolled in the class. The higher the similarity index for a majority of the students, the better it is for the instructor in planning the course. Another type of feedback is to compute the clusters of students who share similar knowledge about topics related to the prerequisites. These clusters help the instructor in identifying the weak areas of the students. The instructor can take appropriate remedial measures by supplementing the lecture material with the required prior topics. This model helps in bridging the information gap between the instructor and the students, thus facilitating a smooth running of the course. WEKA data mining tool is used for building and evaluating the model [6].

#### V. TEXT MINING

In this section, we will review the text mining techniques [4] that aide us in computing the document vector. The first step is to parse the syllabi of the prerequisite courses. In addition to the standard English stop word list, additional technical jargon which does not play a part in the mining process is developed and added to the stop word list. The next phase is stemming and finding equivalent terms used for indexing, counting, and comparing the documents. The remaining words are essentially the concepts and a frequency count is maintained for each of these concepts from all the applicable documents. The first data structure built is the combined documents vector from the prerequisites. This document vector is then compared with the document vector associated with the courses taken by the students. Document similarity computations advocated in the text mining literature (with and without frequency counts) are used to compute the similarity index for each student. At the end of this process, a sorted list of similarity indices of

each student enrolled in the course with reference to the prerequisites is produced.

Once the characteristics of the students were determined based on their prior knowledge, we would like to find similar groups of students sharing common characteristics. Clustering techniques are used to find the similarity measures with the intent that the students within a cluster share the most common knowledge compared to those within any other cluster. These clusters provide the most important feedback to the instructor. The less the number of clusters, and the more the students within each cluster, the better is for the instructor to address the missing knowledge required for the class.

Another data mining technique that can be used in this problem is the classification method [3]. The goal then is to determine if a student meets the prerequisite requirements or not, a binary classification scheme. Based on the past students training data, a decision tree model is built. The current students will then be the test data to determine how many of them meet the requirements or not.

#### VI. PRIVACY ISSUES

To respect the privacy of the students enrolled in the course, the individual student's identities are suppressed in the feedback presented to the instructor. Instead, the students are assigned random identifiers and associated with their profiles. The instructor could share the findings with the students and point out the clusters in the model where prior knowledge is lacking. For each cluster, the knowledge the students have and the knowledge students are deficient in are illustrated.

#### VII. CONCLUSION

In this paper, we presented a framework to capture the students' prior knowledge and share the information with the instructor for the given course. We are in the process of acquiring real data and building the database about the students and their course history. The framework also provides a web interface for instructors at the beginning of each semester to input their syllabus into the system. The text mining capability is being built to compute the document vectors of the course syllabi. Once the model is trained with a good number of students' data, the system could provide immediate feedback to the student upon enrolling in a course how their profile fits with the course requirements.

## APPENDIX A

The Hibernate mappings for the *Semester*, *Student*, *CourseOffering*, *Instructor*, *Course* and *CourseSyllabus* classes are shown below.

### <hibernate-mapping>

```
<class name="Semester" table="SEMESTER">
  <id name="id" type="int" column="SEM_ID">
    <meta attribute="scope-set">protected</meta>
    <generator class="native"/>
  </id>
  <property name="year" type="string" />
  <property name="term" type="string"/>
  <property name="sequence" type="integer"/>
  <!-- bi-directional one-to-many association to
    CourseOffering -->
  <set name="courses" inverse="true" lazy="false"
    cascade="all" >
    <key>
      <column name="SEM_ID" />
    </key>
    <one-to-many class="CourseOffering"/>
  </set>
</class>
<query name="findSemester">
  <![CDATA[
    from Semester as semester
      where upper(semester.term) = upper(:term) and
        semester.year = :year
  ]]>
</query>
```

### </hibernate-mapping>

### <hibernate-mapping>

```
<class name="Student"
  table="STUDENT">
  <id name="id" type="int"
    column="STUDENT_ID">
    <meta attribute="scope-set">protected</meta>
    <generator class="native"/>
  </id>
  <property name="uid" type="string">
    <meta attribute="use-in-tostring">true</meta>
    <column name="UNIVERSITY_ID" not-null="true"
      unique="true" />
  </property>
  <!-- bi-directional many-to-many association to
    CourseOffering -->
  <set name="courses" table="STUDENT_COURSES"
    lazy="false">
    <key column="STUDENT_ID"/>
    <many-to-many class="CourseOffering"
      column="OFFER_ID"/>
  </set>
```

### </hibernate-mapping>

### <hibernate-mapping>

```
<class name="CourseOffering"
  table="COURSEOFFERING">
  <id name="id" type="string" column="OFFER_ID">
    <generator class="uuid" />
  </id>
  <property name="number" type="string" />
  <property name="title" type="string" length="120" />
  <!-- bi-directional many-to-many association to
    Student -->
  <set name="students" table="STUDENT_COURSES"
    inverse="true" lazy="false" cascade="all">
    <key column="OFFER_ID"/>
    <many-to-many class="Student"
      column="STUDENT_ID"/>
  </set>
  <!-- association to Course -->
  <many-to-one name="course" class="Course">
    <column name="COURSE_ID" not-null="true" />
  </many-to-one>
  <!-- association to Semester -->
  <many-to-one name="semester" class="Semester">
    <column name="SEMESTER_ID" not-null="true" />
  </many-to-one>
  <!-- association to Instructor -->
  <many-to-one name="instructor" class="Instructor">
    <column name="INSTRUCTOR_ID"
      not-null="true" />
  </many-to-one>
  <!-- association to CourseSyllabus -->
  <one-to-one name="syllabus"
    class="CourseSyllabus"/>
```

### </class>

### <query name="findOffering">

```
<![CDATA[
  from CourseOffering as offer
    where upper(offer.number) = upper(:number) and
      offer.semester = :semester
  ]]>
</query>
```

### </hibernate-mapping>

### <hibernate-mapping>

```
<class name="Instructor" table="INSTRUCTOR">
  <id name="id" type="int" column="INSTR_ID">
    ... </id>
  <property name="name" type="string"/>
  <!-- association to CourseOffering -->
  <set name="courses" inverse="true" lazy="false"
    cascade="all" >
    <key column name="INSTR_ID" />
    <one-to-many class="CourseOffering"/>
  </set>
</class>
```

## REFERENCES

- [1] J. Elliott, *Hibernate: A Developer's Notebook*, O'Reilly, 2004.
- [2] M. Fowler, "UML Distilled: A Brief Guide to the Standard Object Modeling Language," Addison-Wesley, 2004.
- [3] E. Frank, "Predicting Library of Congress Classifications from Library of Congress Subject Headings," *Journal of the American Society for Information Science and Technology*, 55(3), pp. 214-227.
- [4] E. Loper, "NLTK Tutorial: Text Classification," <http://nltk.sourceforge.net/tutorial/classifying/nochunks.html>, 2005.
- [5] P. Tan, M. Steinbach, and V. Kumar, "Introduction to Data Mining," Addison-Wesley, 2005.
- [6] I. H. Witten and E. Frank, "Data Mining: Practical machine learning tools and techniques," 2nd ed., Morgan Kaufmann, 2005.

```
<query name="instructorByName">
  <![CDATA[
    from Instructor as instructor
    where upper(instructor.name) = upper(:name)
  ]>
</query>
</hibernate-mapping>

<hibernate-mapping>
  <class name="Course" table="COURSE">
    <id name="id" type="int" column="COURSE_ID">
      <meta attribute="scope-set">protected</meta>
      <generator class="native"/>
    </id>
    <property name="number" type="string">
      <column name="NUMBER" not-null="true"
        unique="true" index="COURSE_NUMBER"/>
    </property>
    <property name="name" type="string" />
    <property name="description" type="text" />
    <!-- association to CourseOffering -->
    <set name="offerings" inverse="true" lazy="false"
      cascade="all" >
      <key column name="COURSE_ID" />
      <one-to-many class="CourseOffering"/>
    </set>
    <!-- association to CourseSyllabus -->
    <one-to-one name="syllabus"
      class="CourseSyllabus"/>
  </class>
  <query name="courseByNumber">
    <![CDATA[
      from Course as course
      where upper(course.number) = upper(:number)
    ]>
  </query>
</hibernate-mapping>

<hibernate-mapping>
  <class name="CourseSyllabus"
    table="SYLLABUS">
    <id name="id" type="string" column="SYL_ID">
      <generator class="uuid" />
    </id>
    <version column="revision" name="revision" />
    <property name="overview" type="text" />
    <property name="prereqs" type="text" />
    <property name="objectives" type="text" />
    <property name="readings" type="text" />
    <property name="content" type="text" />
    <property name="properties" type="serializable" />
  </class>
</hibernate-mapping>
```