Jessica Kunkel
SWENG 837
Course Project: Course Scheduling System

Problem Statement and Requirements

The Course Scheduling System (CSS) aims to provide an effective means for students to build their course schedules and enroll in courses for the following semester. The CSS aims to allow professors to select the course sections that fit in their teaching schedule. The CSS also aims to allow Course Administrators to manipulate course data, add and remove courses, and provide exceptions to student enrollment rules where necessary.

Functionalities

The Course Scheduling System needs to provide the following functionalities:

- The CSS shall allow students to schedule classes through an online platform.
- The CSS shall not allow students to schedule more than one class in the same time slot.
- The CSS shall not allow students to enroll in more than 18 credits per semester.
- The CSS shall not allow students to enroll in courses for which they do not meet the prerequisites.
- The CSS shall provide a waitlist function for courses that are at capacity.
- The CSS shall assign courses to appropriately sized classroom spaces.
- The CSS shall not assign more than one class to the same classroom in the same time slot.
- The CSS shall allow professors to register themselves to teach courses.
- The CSS shall not allow professors to register themselves for more than one class in the same time slot.
- The CSS shall allow course administrators to override credit and prerequisite rules for student enrollment.
- The CSS shall allow course administrators to add, remove, and change course data as necessary.

Users

Type	Actor	Goal Description
Primary	Student	• Find classes • Build a desirable schedule • Enroll in courses for the following semester to meet graduation requirements
	Professor	• Manage course offerings, including setting their schedule • Manage grades to track student performance
	Course Administrator	• Override student scheduling rules as needed • Manually enroll students in courses as needed • Manually adjust professor and classroom assignments
Supporting	System Administrator	• Ensure functionality and availability of the CSS • Perform system maintenance • Ensure security of the CSS
	Registrar	• Maintain and update the official course catalog • Oversee the registration process
Offstage	Academic Advisors	• Assist students in academic planning and track students' academic progress

Business Goals

The Course Scheduling System should support the following business goals:

- Reduce the need for error resolution by course administrators by 25%.
    - Includes errors such as student scheduling issues, room scheduling issues, etc.
- Enhance resource utilization by optimizing classroom scheduling and instructor availability.
- Facilitate effective academic advising, reducing reported issues with meeting academic requirements.
- Simplify the course enrollment process for students and professors.

Non-Functional Requirements

Performance

- The CSS shall be able to scale to support over 30,000 simultaneous users.
- 95% of interactions shall be processed in less than one second.
- The CSS shall have a reported uptime of 99.999% (5 Nines)

Security

- The CSS shall utilize data encryption and authorization measures to protect user data.
- The CSS shall provide permission and access controls for all users.
- The CSS shall utilize authentication mechanisms for all users to ensure that malicious agents cannot gain access to the system or its data.

Maintainability

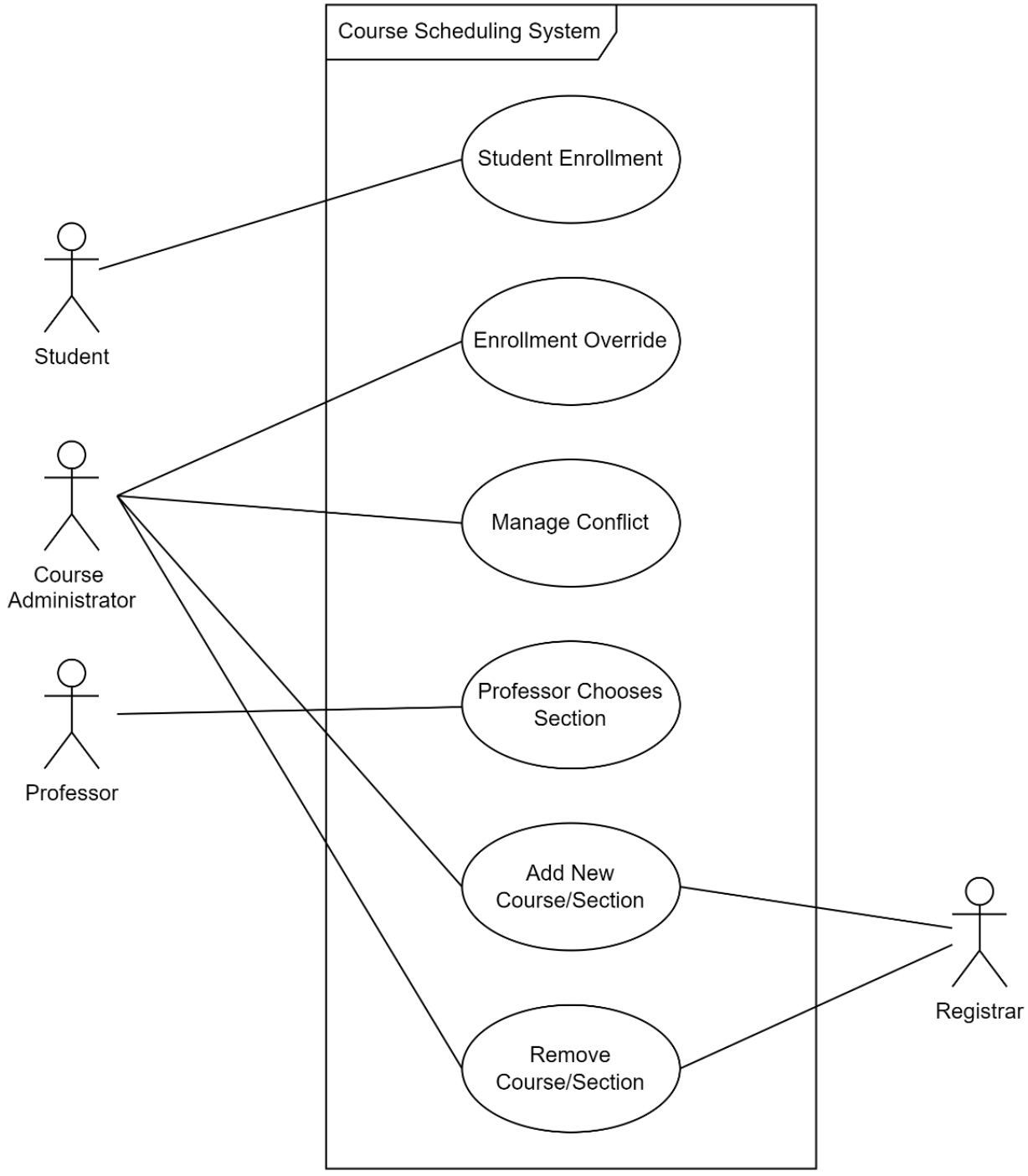
- The system must be able to interface with other university systems such as bursar records and student performance records.
- Code must be well-documented such that future engineers can understand its functionality with ease
- The system must be tested for security (e.g., penetration testing, fuzz testing) as well as for functionality (e.g., unit, integration, and system testing).

Additional Non-functional Requirements

- The system shall include help functionality associated with all user actions.
- The CSS shall be accessible on standard and mobile screen sizes.
- The CSS shall utilize distinguishable colors and displays for individuals with visual impairments.
- The CSS shall allow users to continue activities after re-authentication without loss of data.

Use Cases

Use Case Diagram

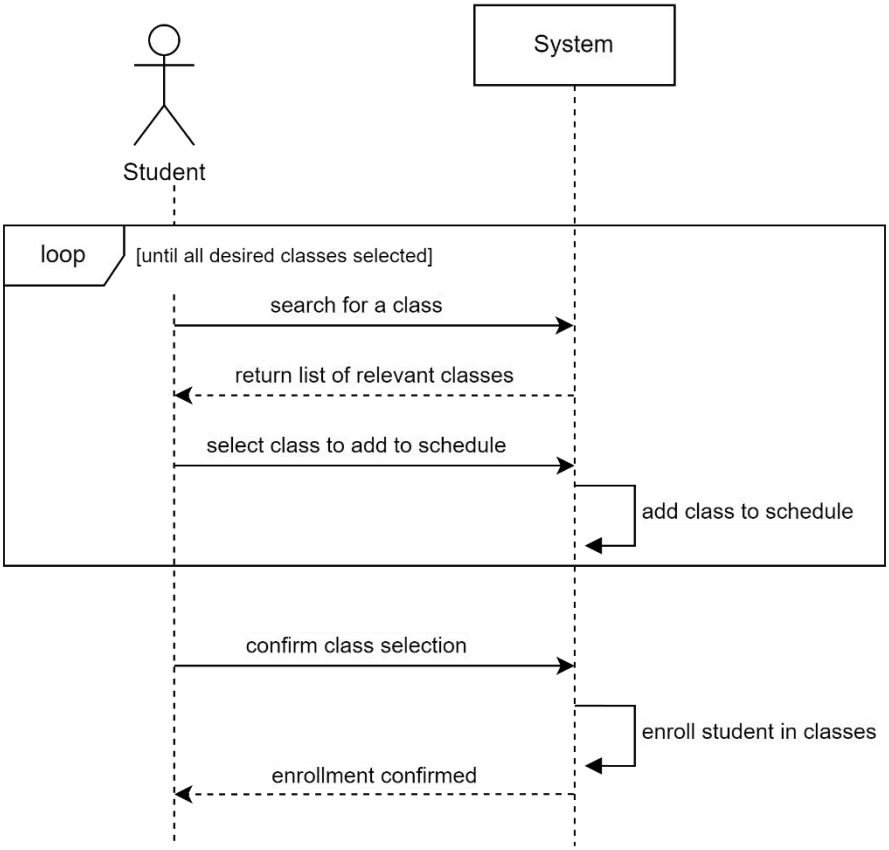


The primary actors in the Course Scheduling System are Students, Professors, and Course Administrators. Students participate in the "Student Enrollment" use case. Professors participate in the "Professor Chooses Section" use case. Course Administrators participate in the "Enrollment Override," "Manage Conflict," "Add New Course/Section," and "Remove Course/Section" use cases. The secondary actor Registrar provides information for the "Add New Course/Section" and "Remove Course/Section" use cases.

Student Enrollment

Use Case Name	Student Enrollment
Scope	Course Scheduling System
Level	User Goal
Primary Actor	Student
Stakeholders and Interests	Student: wants to find classes, enroll in classes, and fulfill graduation requirements Course Administrator: wants course scheduling to proceed smoothly without the need to provide manual enrollment overrides Academic Advisor: wants students to meet graduation requirements and achieve degree progress Registrar: wants course scheduling to proceed smoothly without need for intervention by the Course or System Administrators
Preconditions	• Courses are available in the scheduling system • The Student has been authenticated
Success Guarantee	The Student is able to find, select, and enroll in their desired classes.
Main Success Scenario	1. Student searches for a class 2. Student selects one section of the class to add to their schedule 3. Class is added to schedule *Steps 1-3 are repeated until the student has added all desired classes to their schedule.* 4. Student confirms class selections 5. System enrolls students in their selected classes
Extensions	3a. Student selects a class whose time slot overlaps with one already on their schedule 1. The system notifies the student of the scheduling conflict and will not allow the student to continue to confirmation 2. The student chooses a different section of the course, or a different class 3b. Student selects a class for which they do not meet the prerequisite requirements 1. The system notifies the student of the prerequisite issue and will not allow the student to continue to confirmation 2. The student chooses another course, or requests an override from the Course Administrator 3c. Student exceeds a total of 18 credits on their schedule 1. The system notifies the student of the over-scheduling error and will not allow the student to continue to class confirmation 2. The student removes one or more classes from their schedule or requests and override from the Course Administrator 3d. Course is full 1. The system notifies the student that they will be put on the waitlist for the selected section 2. The student confirms their position on the waitlist or selects an open section of the course
Special Requirements	Usability, Performance, Availability

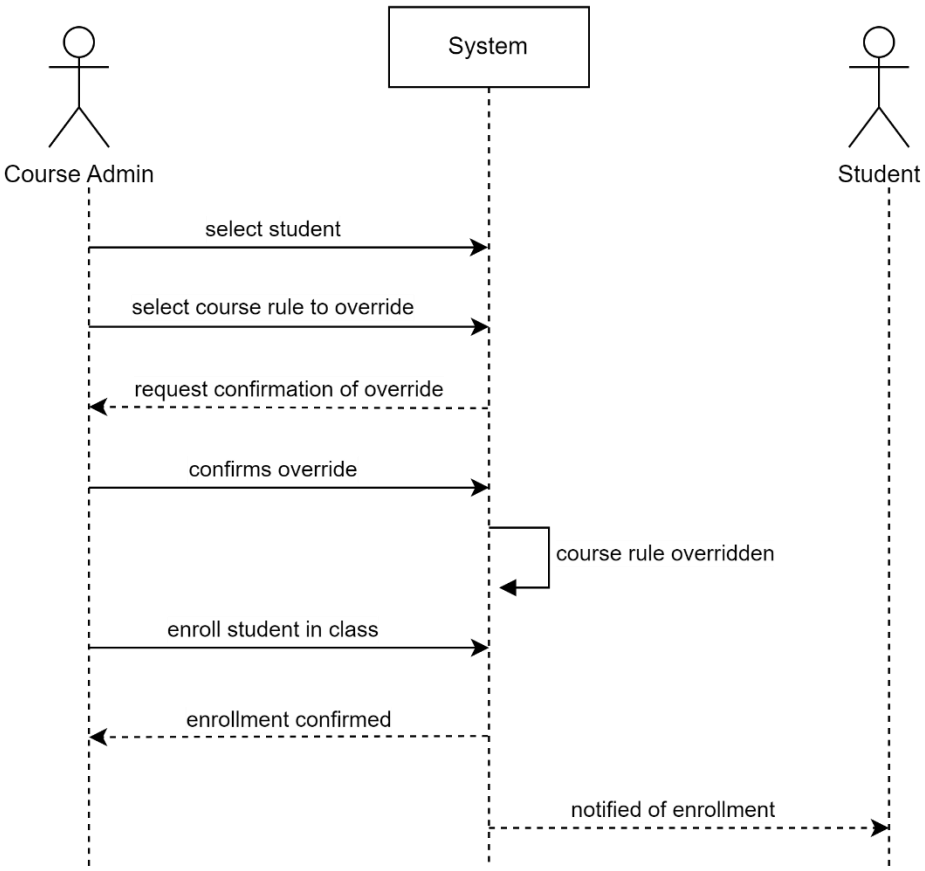
Student Enrollment



Enrollment Override

Use Case Name	Enrollment Override
Scope	Course Scheduling System
Level	User Goal
Primary Actor	Course Administrator
Stakeholders and Interests	Student: wants to enroll in classes that they fulfill graduation requirements Course Administrator: wants to quickly and easily override enrollment rules for a particular student, and no others Academic Advisor: wants students to meet graduation requirements and achieve degree progress Registrar: wants any necessary rule overrides to proceed smoothly without need for intervention by the System Administrators
Preconditions	• Courses are available in the scheduling system • The Course Administrator has been authenticated • The Course Administrator has received a request to override an enrollment rule
Success Guarantee	Course enrollment rule is successfully overridden, and Student can successfully enroll in their desired courses
Main Success Scenario	1. Course Administrator selects the student account from whom they received the override request 2. Course Administrator selects the course rule to override 3. System requests confirmation of the override 4. Course Administrator confirms the override 5. Enrollment rule is overridden in only this case 6. Student is manually added to the class 7. System notifies the Student of the successful enrollment
Extensions	1a. Course Administrator has selected the wrong student account 1. Course Administrator does not find the appropriate course rule to override 2. Course Administrator repeats the student account search and selection 3. Course Administrator selects the appropriate student 4a. Course Administrator has selected the wrong course rule 1. Course Administrator reviews course rule selection in confirmation screen and sees the incorrect rule 2. Course Administrator does not confirm the change 3. Course Administrator undoes the selection and chooses the appropriate rule to override 4. Course Administrator confirms the fixed override 4b. Course Administrator has selected a course rule for which there is no issue 1. System notifies Course Administrator that there is no issue with the selected rule 2. Course Administrator undoes the selection and chooses the appropriate rule to override 3. Course Administrator confirms the fixed override
Special Requirements	Usability, Security, Performance, Availability

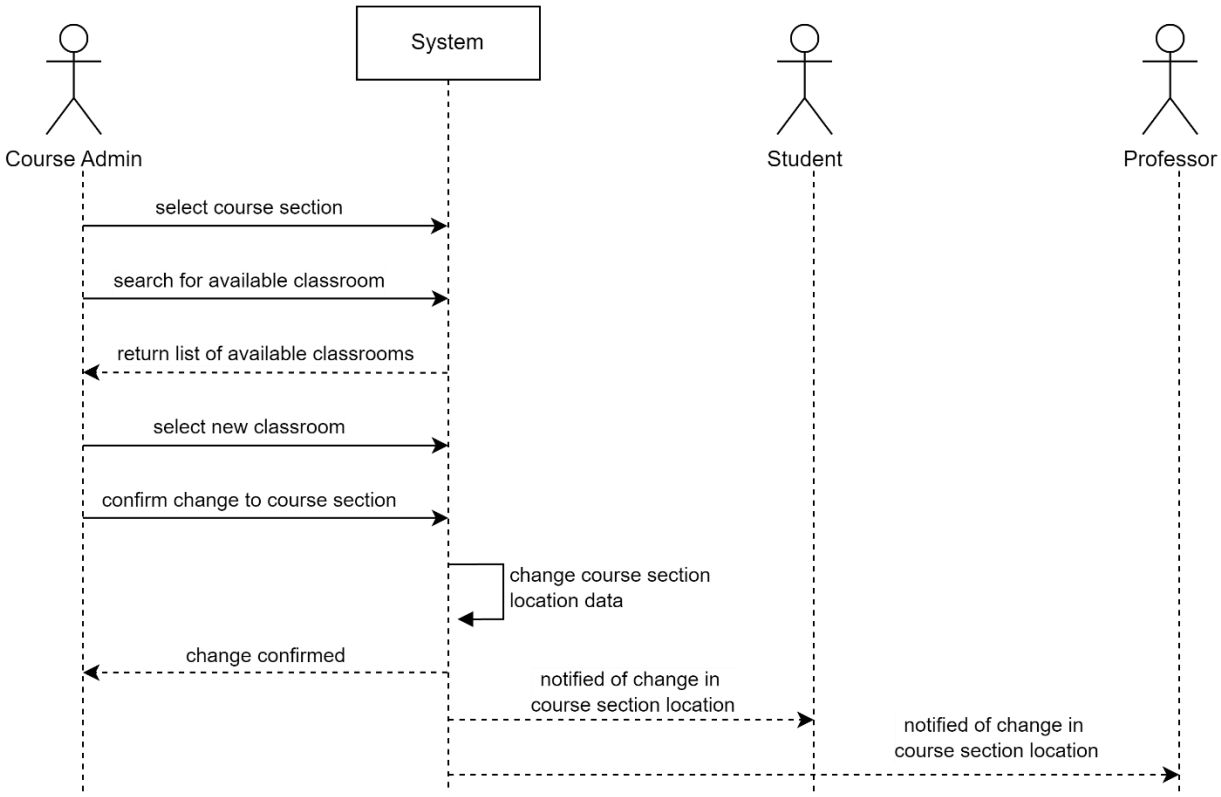
Enrollment Override



Manage Conflict

Use Case Name	Manage Conflict
Scope	Course Scheduling System
Level	User Goal
Primary Actor	Course Administrator
Stakeholders and Interests	Student: wants to find classes, enroll in classes, and fulfill graduation requirements Course Administrator: wants to resolve scheduling conflicts
Preconditions	• Courses are available in the scheduling system • There is a classroom conflict (e.g., class missing a location due to system inability to find an appropriate classroom, more than one class in the same room at the same time due to sys admin error) • The Course Administrator is authenticated
Success Guarantee	All classes have a location No two classes are scheduled for the same location at the same time
Main Success Scenario	1. Course Administrator selects the course section to manage 2. Course Administrator finds a classroom that is unoccupied for the entire time slot 3. Course Administrator confirms section change 4. Section assignment is successfully changed 5. System notifies students enrolled in that section of the change 6. System notifies the professor of that section of the change
Extensions	2a. There are no classrooms available for the entire time slot 1. Course Administrator searches for any location, any time slot of the required length 2. System returns a list of available times and locations 3. Course Administrator selects an available slot 4. Course Administrator confirms change to course section *Returns to success scenario at Step 4* 5b. There has been a change to the time slot of the schedule, instead of or in addition to the location change 1. Students in the affected section have their schedules automatically changed 2. Students are notified of any new conflicts with their existing schedule 3. Students must reorganize their schedule selections to resolve the conflict 6b. There has been a change to the time slot of the schedule, instead of or in addition to the location change 1. The Professor(s) teaching the affected section has their schedule(s) automatically changed 2. The Professor is notified of any new conflicts with their existing schedule 3. The Professor must reorganize their schedule selections to resolve the conflict
Special Requirements	Usability, Security, Performance, Availability

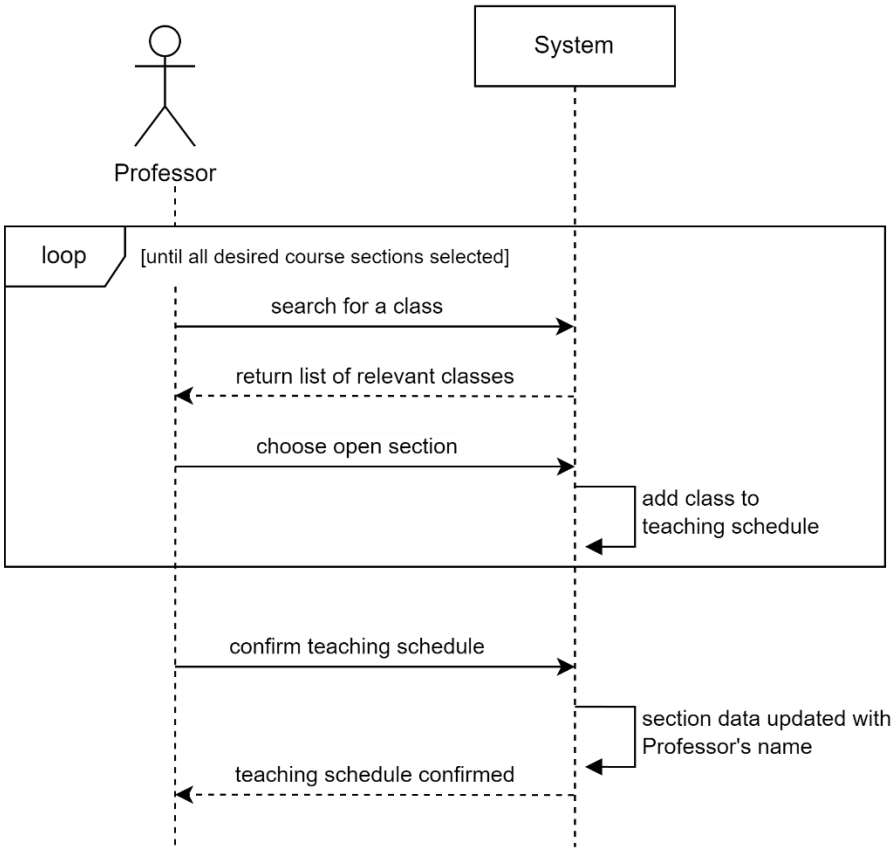
Manage Conflict



Professor Chooses Section

Use Case Name	Professor Chooses Section
Scope	Course Scheduling System
Level	User Goal
Primary Actor	Professor
Stakeholders and Interests	Professor: wants to choose courses and sections to teach, building a schedule and workload that works for them Course Administrator: wants course scheduling to proceed smoothly without the need to provide manual enrollment overrides Registrar: wants course scheduling to proceed smoothly without need for intervention by the Course or System Administrators
Preconditions	• Courses are available in the scheduling system • The Professor has been authenticated • Generally occurs prior to student course selection
Success Guarantee	The Professor is able to find, select, and assign themselves to the course section(s) they will teach
Main Success Scenario	1. Professor searches for a class 2. Professor chooses an open section (time and location) to teach 3. Class is added to teaching schedule *Steps 1-3 are repeated until the Professor has added all desired classes to their schedule.* 4. Professor confirms teaching schedule 5. Section data is updated with the Professor's name
Extensions	2a. Professor selects a course they are not qualified to teach 1. System flags the course in the teaching schedule and confirmation page 2. System will not allow the Professor to proceed with confirmation without Course Administrator approval 3. Professor contacts Course Administrator or removes course section and restarts at step 1 of main success scenario. 5a. Professor chooses a course after students have begun enrollment 1. System notifies students enrolled in the section of the change in professor
Special Requirements	Usability, Performance, Availability

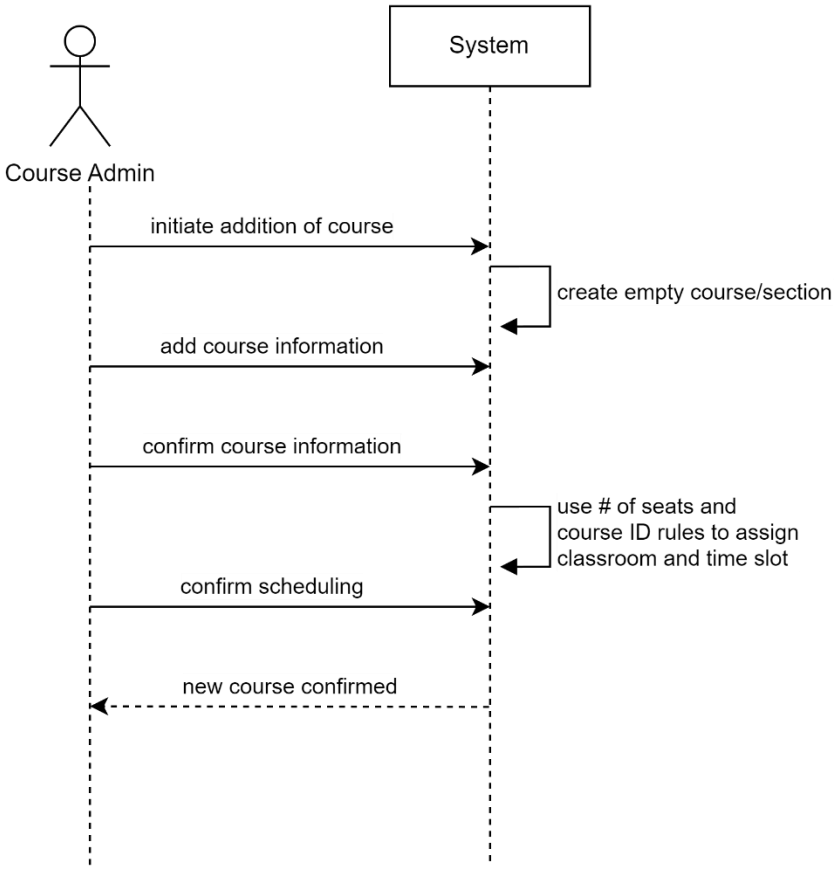
Professor chooses section



Add New Course/Section

Use Case Name	Add New Course/Section
Scope	Course Scheduling System
Level	User Goal
Primary Actor	Course Administrator
Stakeholders and Interests	Course Administrator: wants to successfully add a new course or section to the system Registrar: wants course addition to be accurate
Preconditions	• The Registrar has added course information to the catalog (information source outside the CSS) – OR – Registrar has indicated the need for an additional section of a course • The Course Administrator has been authenticated
Success Guarantee	A new course or section of a course is successfully added to the system
Main Success Scenario	1. Course Administrator initiates addition of a course or section 2. Course Administrator adds information about the course provided by Registrar: course ID, description, required number of seats, number of credits, prerequisites, number of sections, etc. 3. Course Administrator confirms course information 4. System uses number of seats and course ID rules to assign the class to a classroom and time slot 5. Course Administrator confirms scheduling
Extensions	2a. Course Administrator includes incorrect information 1. Course Administrator does not confirm course information 2. Course Administrator updates information appropriately 4a. System cannot find a classroom using number of seats and course ID rules 1. <<include>> Manage Conflict
Special Requirements	Usability, Performance, Availability

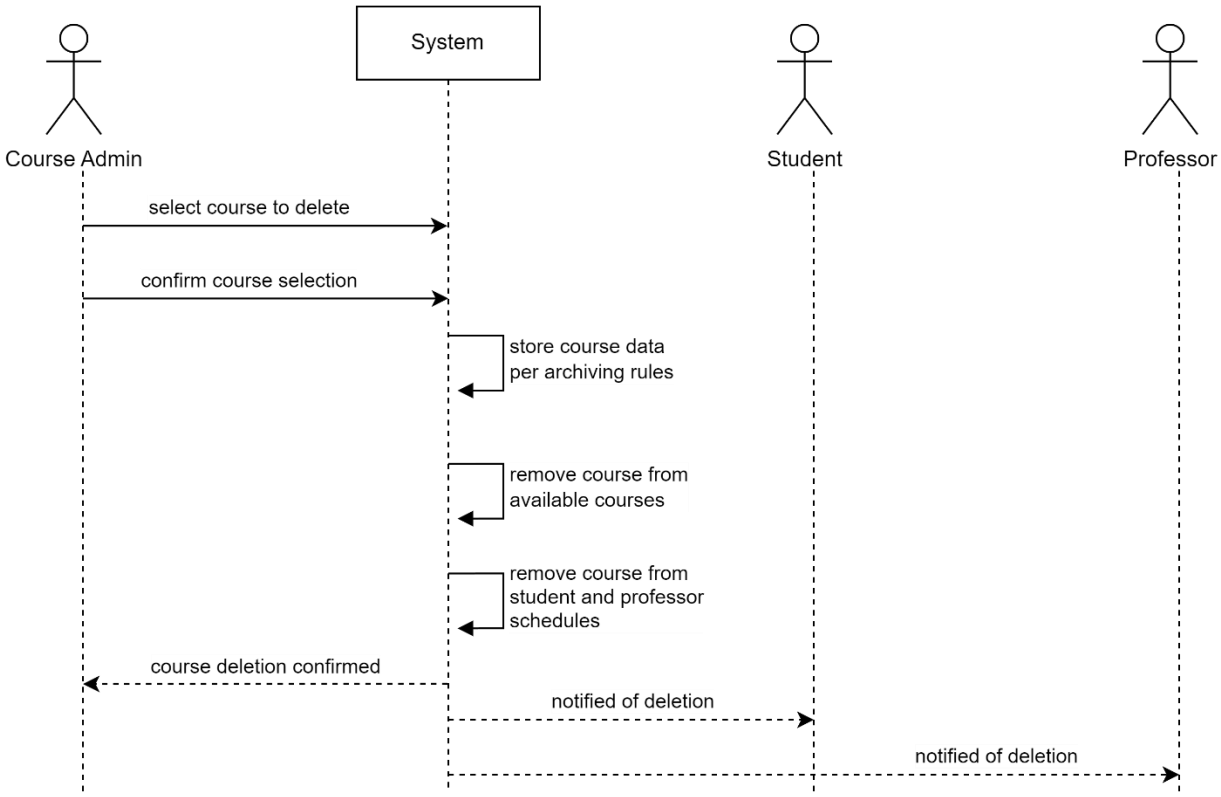
Add new course/section



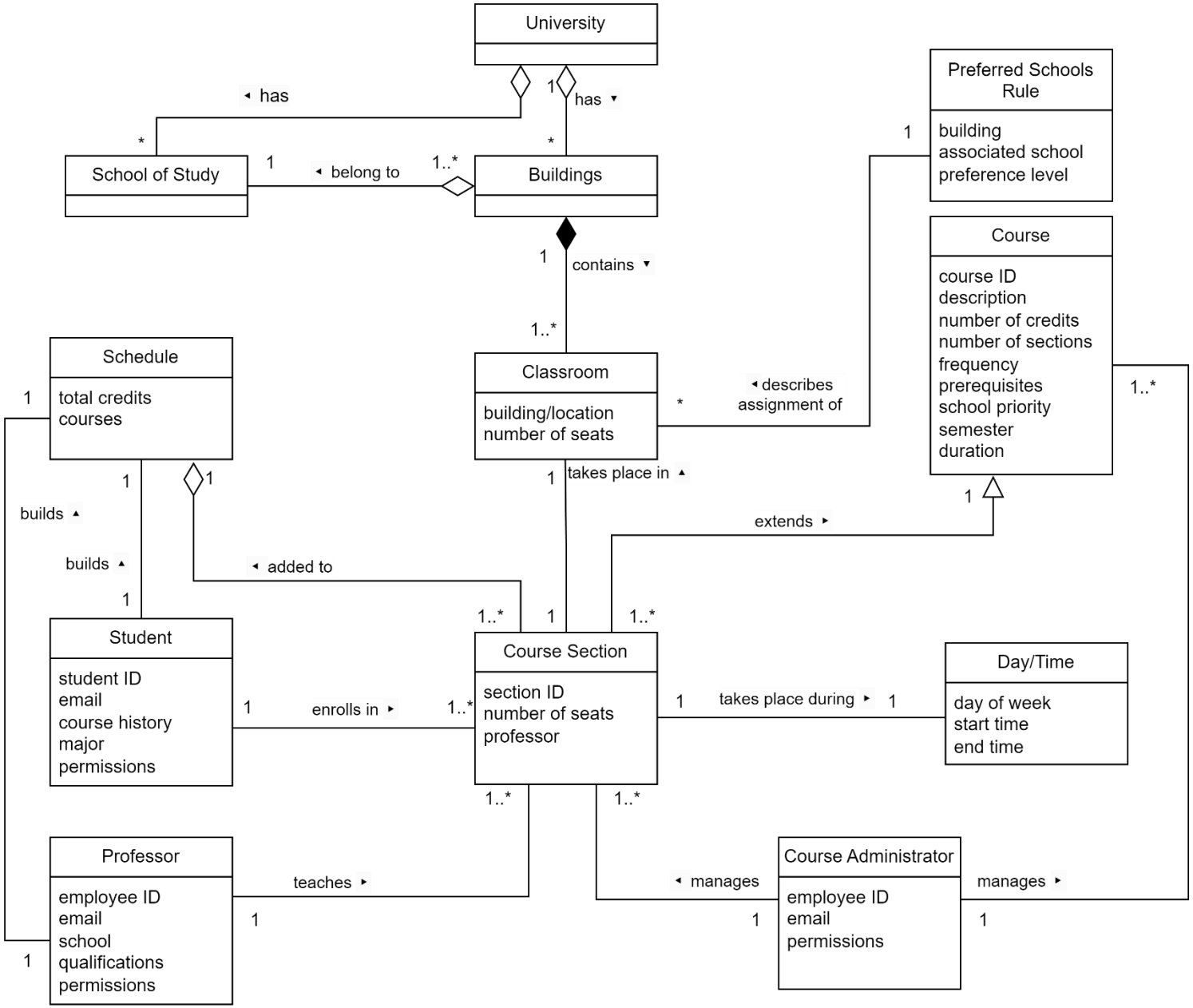
Remove Course/Section

Use Case Name	Remove Course/Section
Scope	Course Scheduling System
Level	User Goal
Primary Actor	Course Administrator
Stakeholders and Interests	Course Administrator: wants to successfully remove a course or section Registrar: wants course deletion to be accurate
Preconditions	• The Registrar has canceled a course (e.g., lack of student enrollment, lack of staff) – OR – Registrar has indicated the need to remove an unneeded section of a course • The Course Administrator has been authenticated
Success Guarantee	A course or section of a course is removed from enrollment possibilities. Course/section data is saved for any potential future need.
Main Success Scenario	1. Course Administrator selects a course or section to delete 2. Course Administrator confirms the course or section being deleted 3. System stores course/section data per course archiving rules 4. System removes the course/section from available courses 5. System removes the course/section from the schedules of any enrolled students and professors 6. System notifies students and professors of deletion and need for attention
Extensions	1a. Course Administrator chooses the wrong course 1. Course Administrator does not confirm course deletion 2. Course Administrator is taken back to step 1 of main success scenario
Special Requirements	Usability, Performance, Availability

Remove Course/Section



Domain Model



The Domain Model begins at the University level. One University has many Buildings and Schools of Study. One to many Buildings belong to one School of Study. One Building contains one to many Classrooms, with attributes like building/location and number of seats. The Preferred Schools Rule has attributes like building, associated school, and preference level, and describes the assignment of Classrooms.

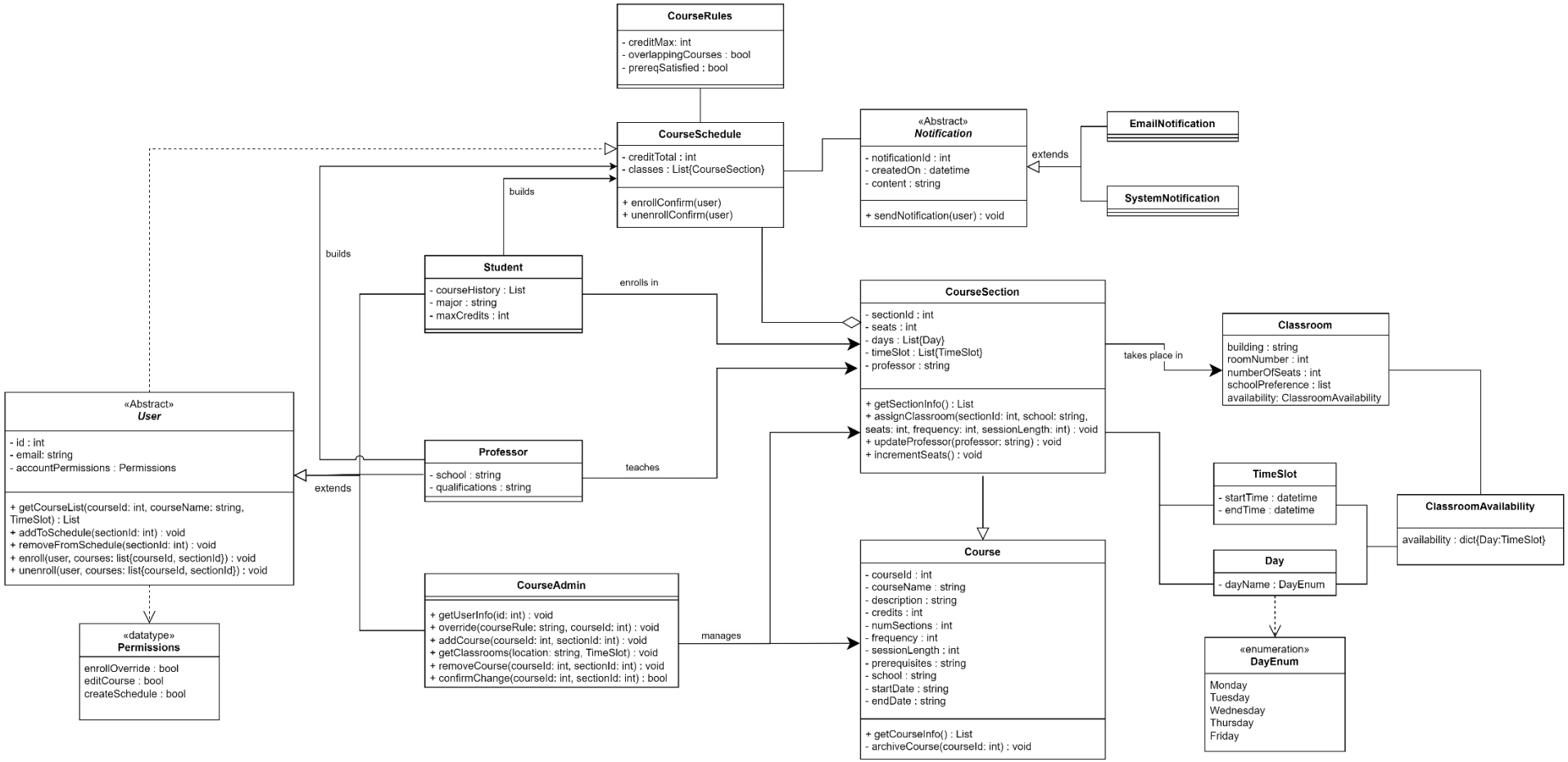
Course has attributes like course ID, description, number of credits, number of sections, frequency, prerequisites, school priority, semester, and duration. One Course is extended by one to many Course Sections. Course Section has attributes like section ID, number of seats, and professor, and takes place during a Day/Time (described by day of week, start time, and end time). Course Section is added to Schedule.

Student, with attributes like student ID, email, course history, major, and permissions, enrolls in one to many Course Sections. One Student builds one Schedule.

Professor, with attributes like employee ID, email, school, qualifications, and permissions, teaches one to many Course Sections. One Professor builds one Schedule.

Course Administrator, with attributes like employee ID, email, and permissions, manages one to many Courses and one to many Course Sections.

Design Class Diagram



The Class Diagram is derived from and extends the Domain Model.

The Abstract User class generalizes the use classes, pulling out common attributes like ID, email, and account permissions. These account permissions have their own datatype of boolean expressions describing whether users can override enrollment, edit courses, or create schedules. In general, Student and Professor classes have the "Create Schedule" permission, while Course Administrators have the "Enroll Override" and "Edit Course" permissions. The User class has methods that allow users to search for courses, add and remove courses from schedules, and enroll and unenroll in courses. Student extends User, with additional attributes like course history, major, and maximum number of credits. Professor extends User, with additional attributes like school and qualifications. Course Administrator extends User and has additional methods that allow the course admin to override enrollment (get user information, override), create new courses, remove courses, manage conflicts (get classrooms, as a search for an available location and time), and confirm all changes being made. The User class as a whole is dependent on Course Schedules.

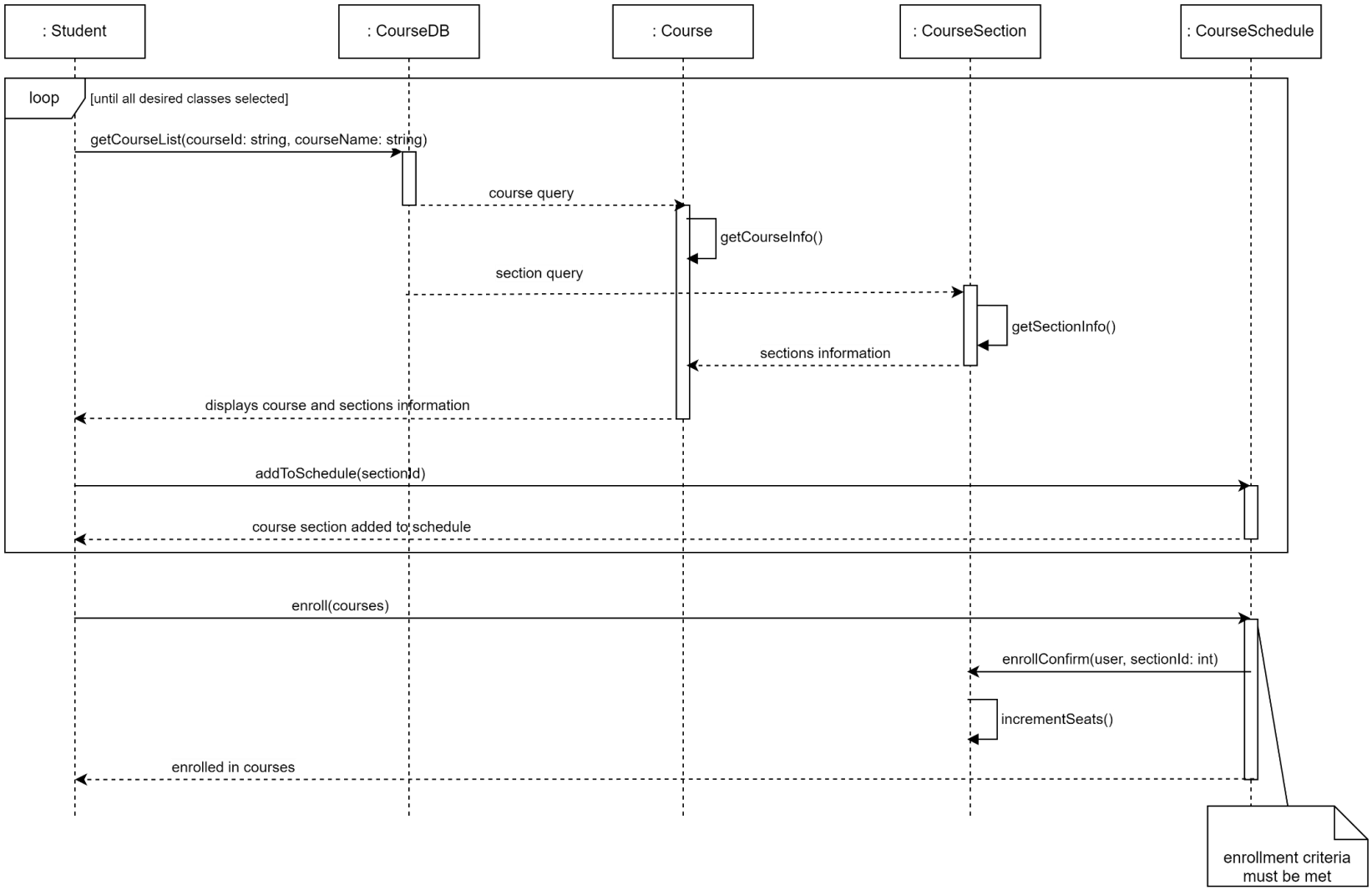
Course has attributes like course ID, name, description, number of credits, number of sections, frequency of occurrence (how many sessions in a week), session length (the amount of time a session takes, e.g., 50 minutes), prerequisites, school (which school in the university the course is a part of), start date, and end date. Course has a public method to return all of its information and attributes, and a private method (archiveCourse) to send all its data to an archive database so that the information isn't lost completely, but also isn't able to be enrolled in.

Course Section extends Course, with additional attributes section ID, number of seats, days (days of the week the course section takes place on), time slot (a list of the times at which the course section takes place), and professor (that teaches the course section). Course Section inherits the getCourseInfo method, and has its own public methods getSectionInfo (returns section-specific information), assignClassroom (a course section assigns itself to a classroom upon creation), and updateProfessor (when a professor registers themself to teach a course section, it updates the course section's professor attribute). Course Sections take place in classrooms, given the classroom's availability.

Students and professors add course sections to their course schedules. Course Schedule has attributes describing the total number of credits and the list of classes it contains. Whenever there are changes to courses and course sections that are part of course schedules, an email and/or system notification is sent to affected users. Course Schedule is associated with Course Rules, which are used to evaluate the validity of the schedule.The classes TimeSlot and Day are used to describe course sections and classroom availabilities, and the dayName attribute uses an enumeration of days of the week that courses can take place on.

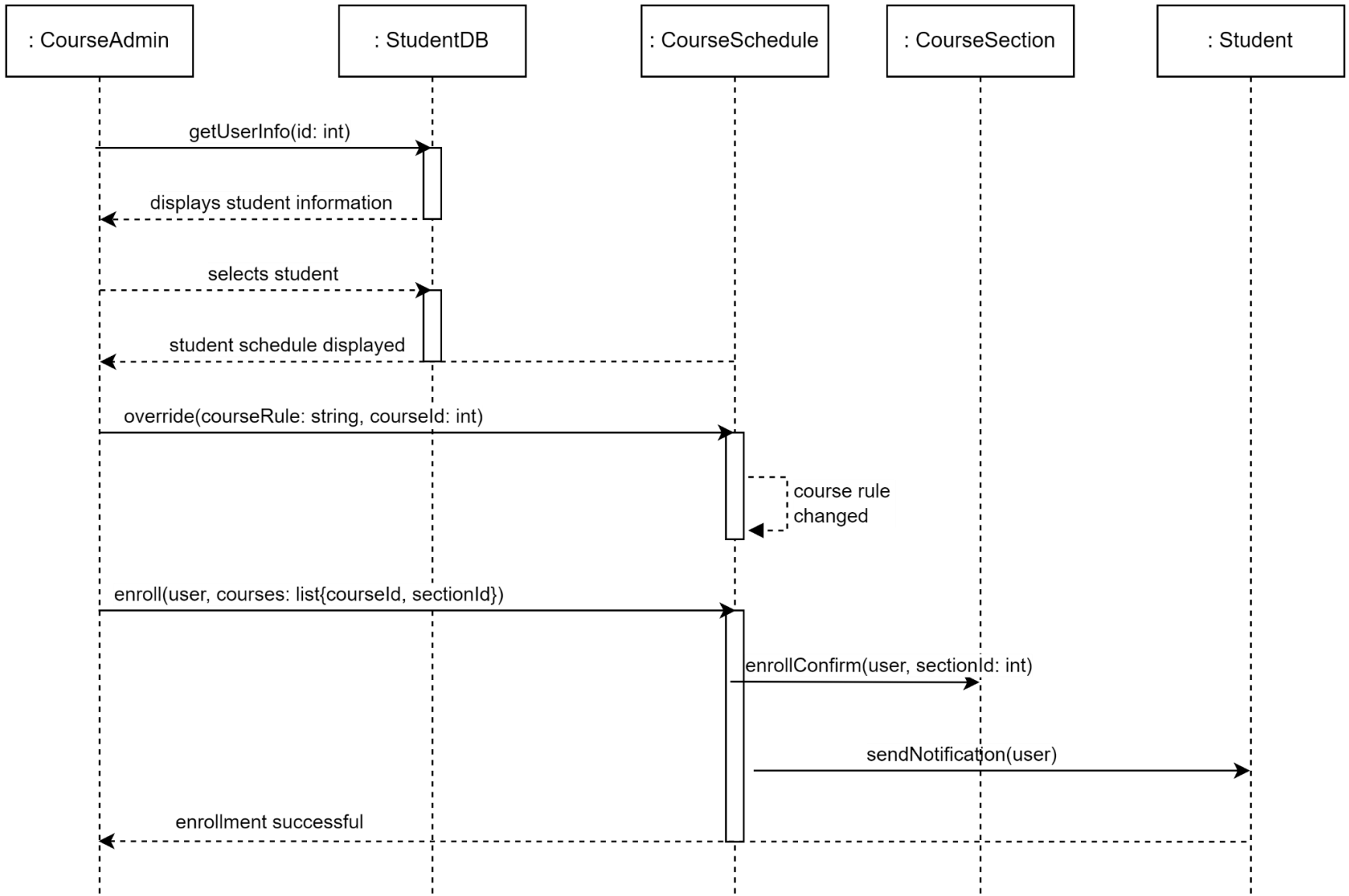
Sequence Diagrams

Student Enrollment



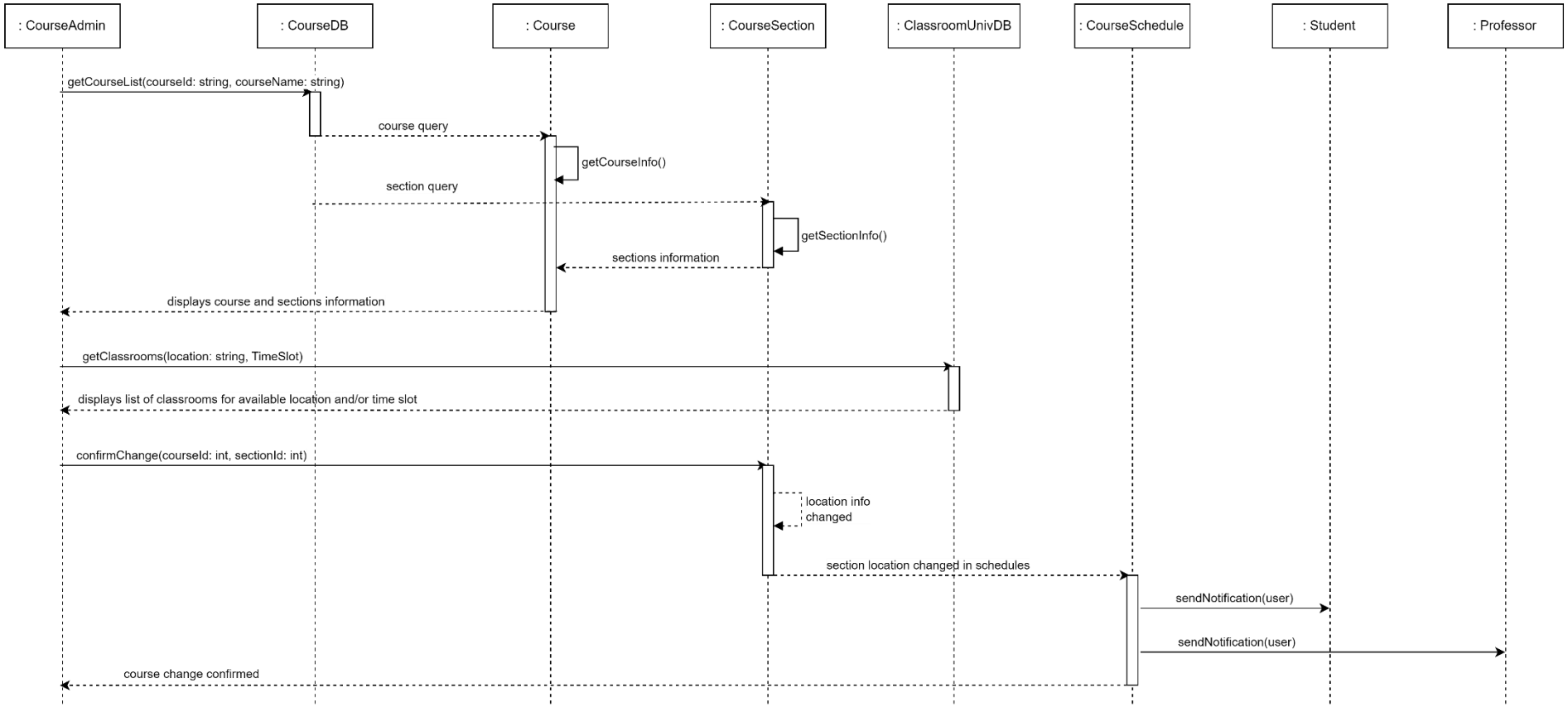
The student searches for a course, invoking getCourseList to the Course Database, which gets the course info and section info and displays the information to the student. The student adds a selected course to their schedule, and this process repeats until the student has selected all the courses they want. The student enrolls in the classes in their schedule, and if all of the enrollment criteria are met, the course schedule confirms enrollment of the student, invoking incrementSeats on the course section and reducing the number of available seats.

Enrollment Override



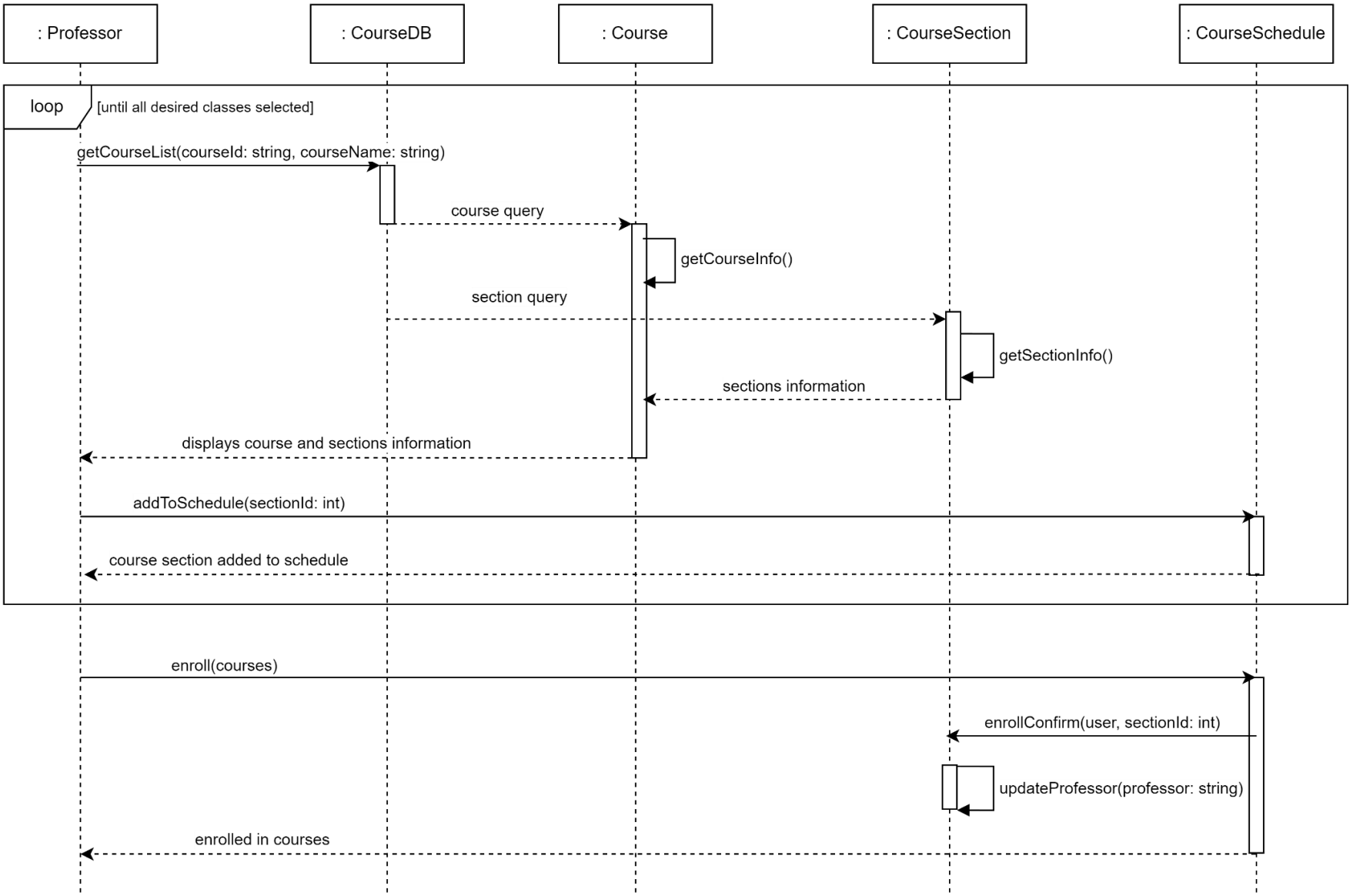
The course administrator searches for student information in the student database, and selects a student. The student's course schedule, from the course schedule object through the student database, is displayed to the course administrator. The course administrator overrides the course rule, changing the rule in question. The course admin then enrolls the student in the course and the schedule confirms enrollment of the student, invoking incrementSeats on the course section and reducing the number of available seats. The course schedule notifies the student of the change.

Manage Conflict



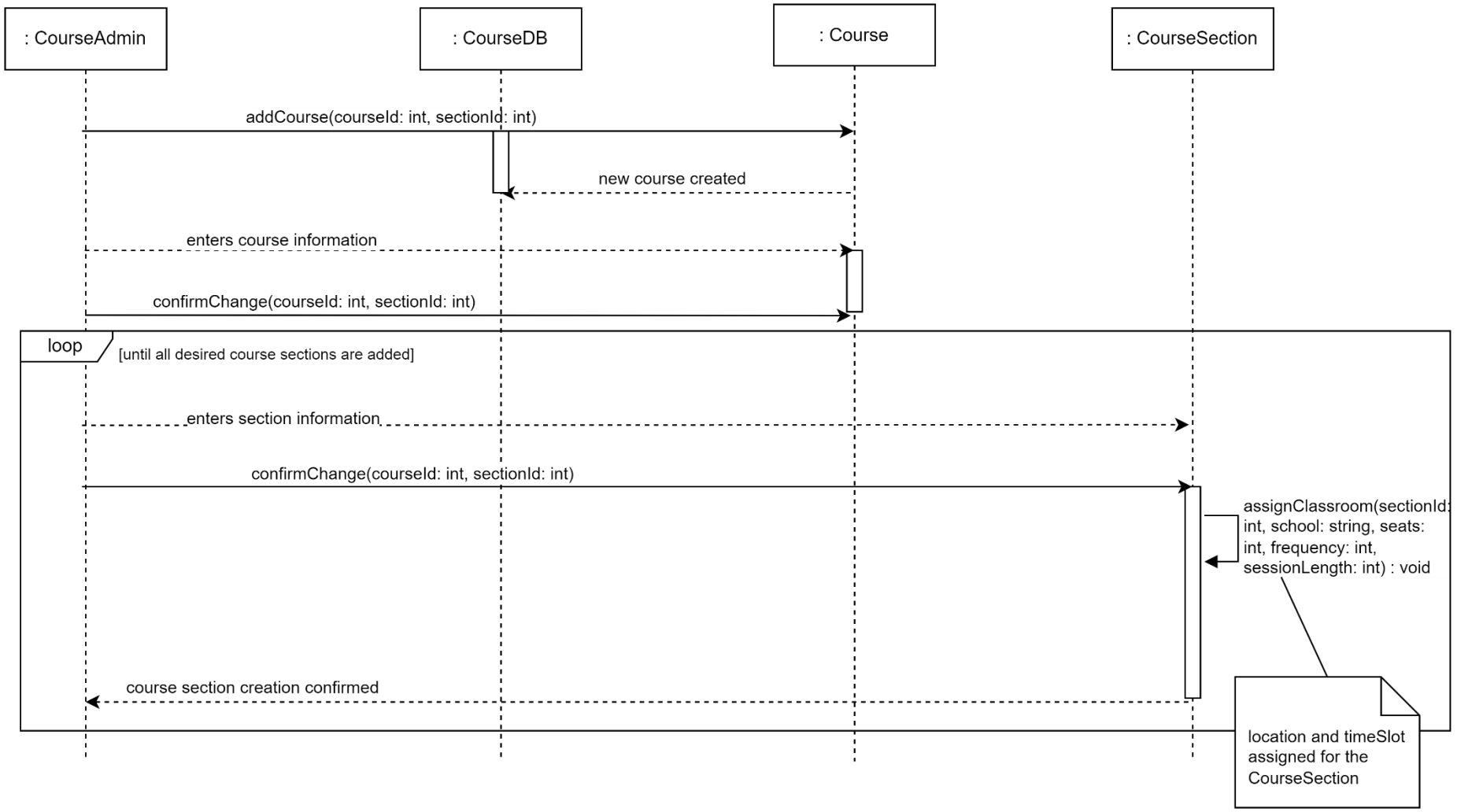
The course administrator searches for a course, invoking getCourseList to the Course Database, which gets the course info and section info and displays the information to the course administrator. Separately, the course administrator searches for classrooms through the Classroom University Database to find an available location and time for the course section. The course admin makes the changes to the location and or time of the course section, and the section information is changed in any affected course schedules. The students and professors with the changed course section in their schedules are notified of the change.

Professor Chooses Section



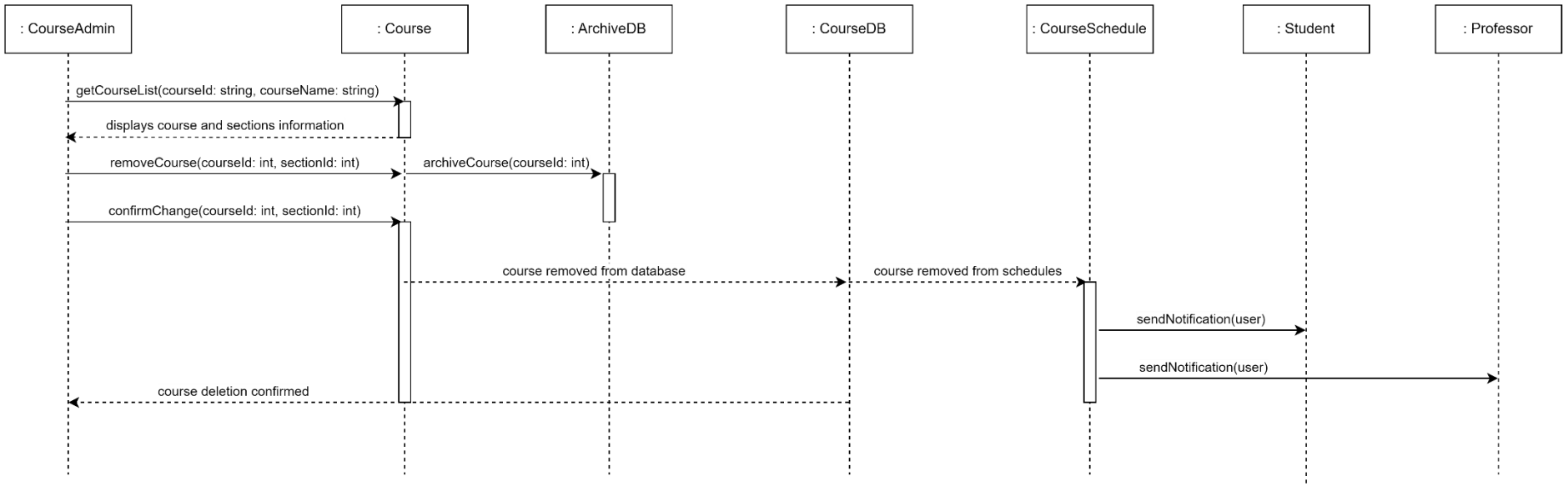
The professor searches for a course, invoking getCourseList to the Course Database, which gets the course info and section info and displays the information to the professor. The professor adds a selected course to their schedule, and this process repeats until the student has selected all the courses they want. The professor enrolls in the classes in their schedule, and if no course sections overlap in their occurrence, the course schedule confirms enrollment of the professor, invoking updateProfessor on the course section and updating the course section's professor attribute.

Add New Course/Section



The course administrator invokes addCourse to create a new course object. The course admin enters course information and confirms those changes. The professor enters course section information and confirms those changes. The course section assigns itself to a classroom, and a location and time slot is assigned to the course section. This course section process repeats until all course sections are added.

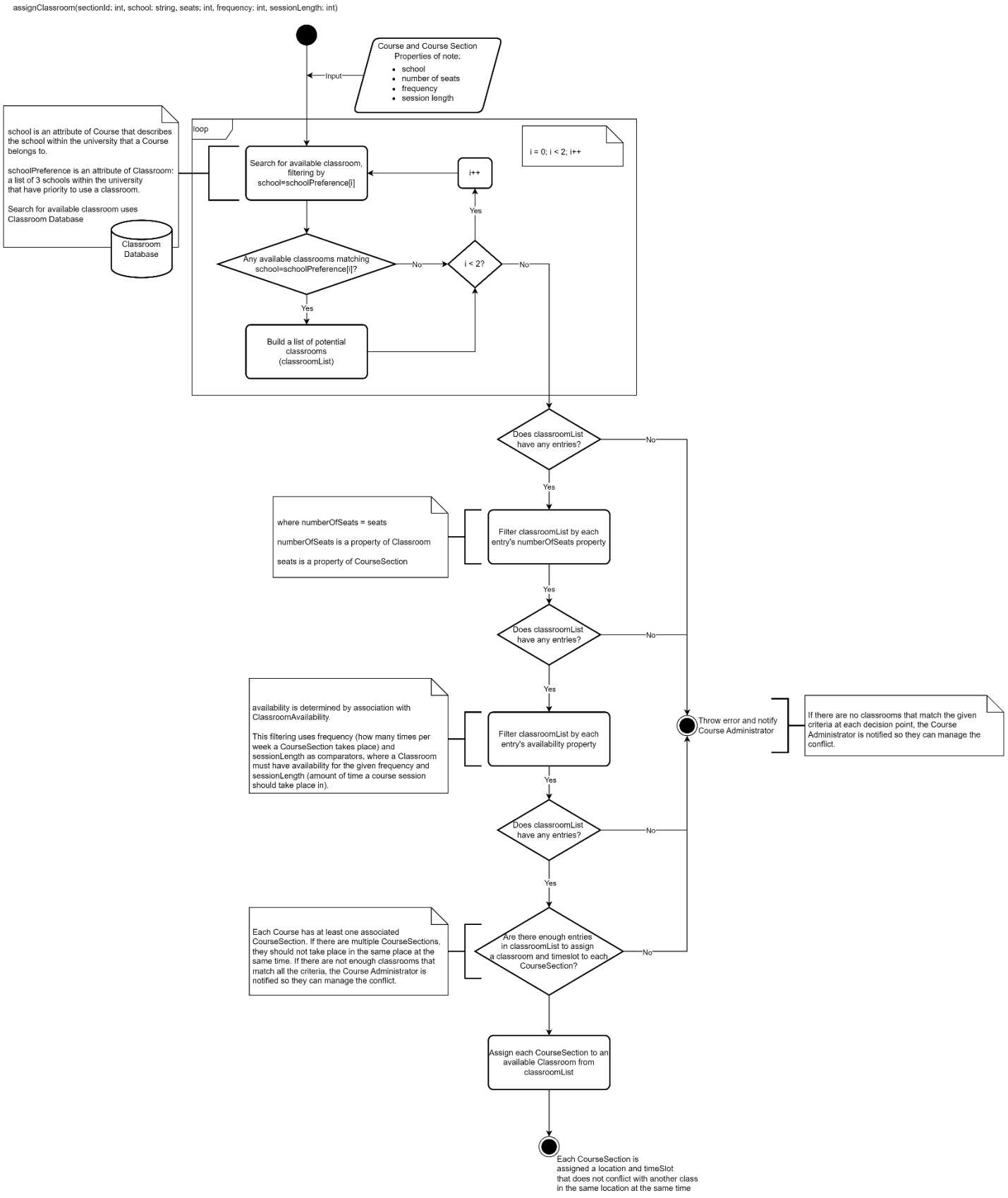
Remove Course/Section



The course administrator searches for a course, invoking getCourseList to the Course Database, which gets the course info and section info and displays the information to the course administrator. The course administrator invokes removeCourse, and the Course invokes archiveCourse to send the course data to the Archive database. The course admin confirms the removal of the course. The course is then removed from the course database and from course schedules. Course Schedule sends notifications of the course removal to any affected students and professors. The deletion of the course is confirmed to the admin.

Activity Diagram

assignClassroom(sectionId: int, school: string, seats: int, frequency: int, sessionLength: int)



The activity diagram shows how the assignClassroom method works.

There is an input to the method of the course section's school, number of seats, frequency, and session length. The system searches for an available classroom utilizing the classroom database, filtering by the course section's school matching the classroom's school preference. A classroom has a list of 3 schools within the university that have priority to use a classroom (schoolPriority). The system checks whether any available classrooms' school preference match the school, and if so the classroom is added to a list of potential classrooms. If not, the system checks the iteration of the loop and iterates if i<2. If the classroom is added to a list of potential classrooms, the system checks the iteration (i<2?) and either iterates (i<2) or moves to the next decision point (i>=2). This loop occurs three times, one for each index of schoolPreference.

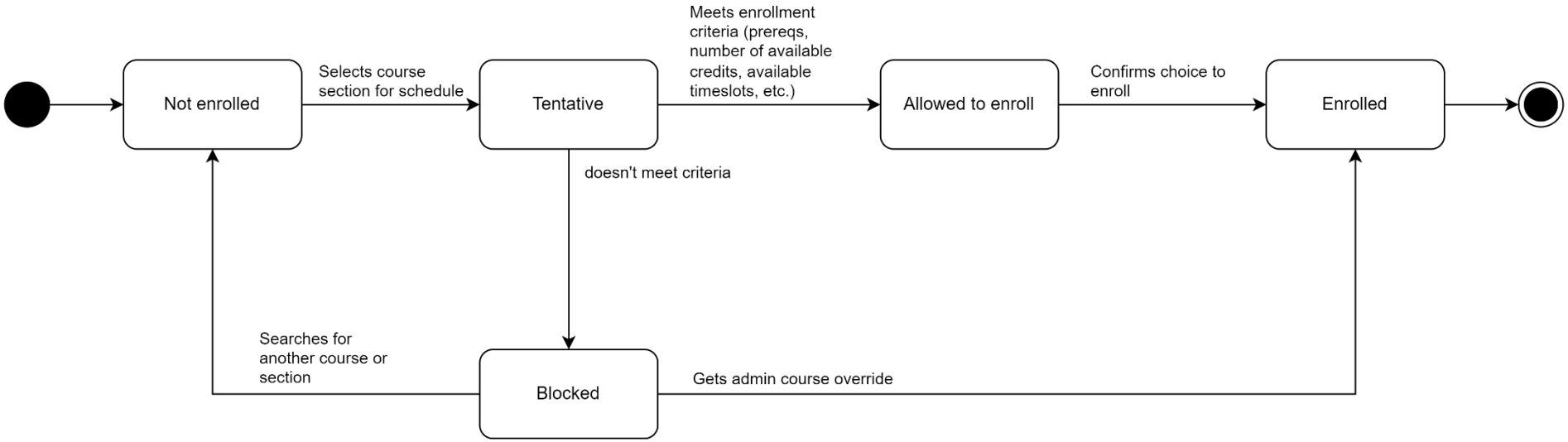
At each step hereafter, the system checks if the list of classrooms has any entries, and applies another filter to the list. If at any point the list of potential classrooms doesn't have any entries, the system throws an error and notifies the course administrator so they can manage the conflict.

First, the system filters the list of classrooms by the number of seats, where the number of seats in a classroom must be greater than or equal to the number of seats in the course section. Then, the system filters by the availability of the classroom. This filtering uses frequency (how many times per week a CourseSection takes place) and sessionLength as comparators, where a Classroom must have availability for the given frequency and sessionLength (amount of time a course session should take place in.

Finally, the system checks to see if there are enough classrooms in the list to assign a classroom and time slot to each course section. If all steps are successful, each course section is assigned a classroom (location and time slot) that doesn't conflict with another class in the same location at the same time.

State Diagram

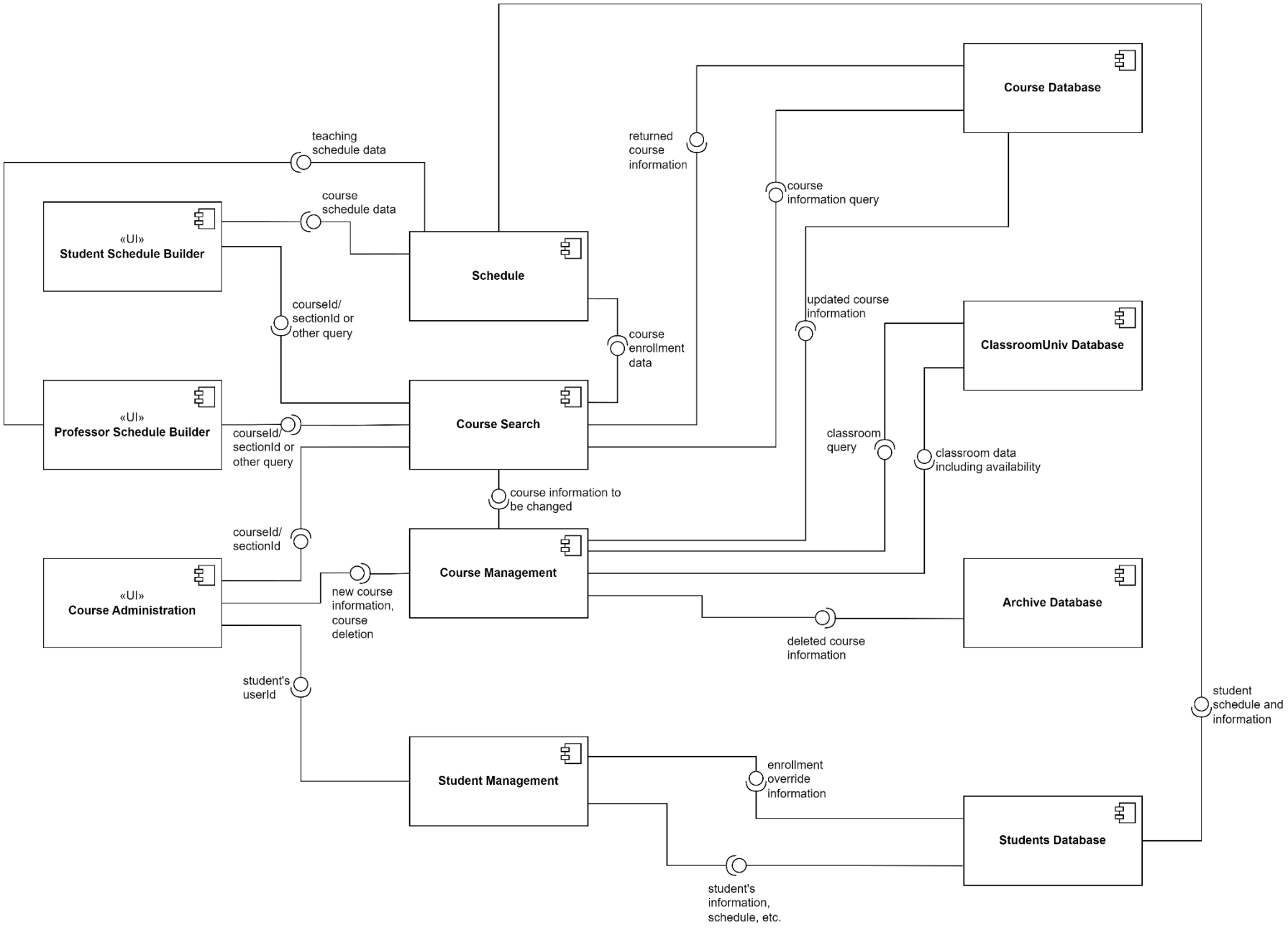
Student Object



The state diagram shows the potential states of a Student object.

The Student begins as "Not enrolled." When a course is selected for the schedule, the Student is "Tentative" (tentatively enrolled). The system checks if the Student meets the enrollment criteria. If the student meets the criteria, Student is "Allowed to Enroll," but if the criteria are not met the student is "Blocked." From "Blocked" a student can search for another course or section and become "Not enrolled," or the student can get an administrator override and become "Enrolled." From "Allowed to Enroll," the student confirms their choice to enroll in courses and becomes "Enrolled," and the state machine ends.

Component Diagram



The component diagram shows the data flow and requirements between user interfaces for each user type, major components, and information stores (database).

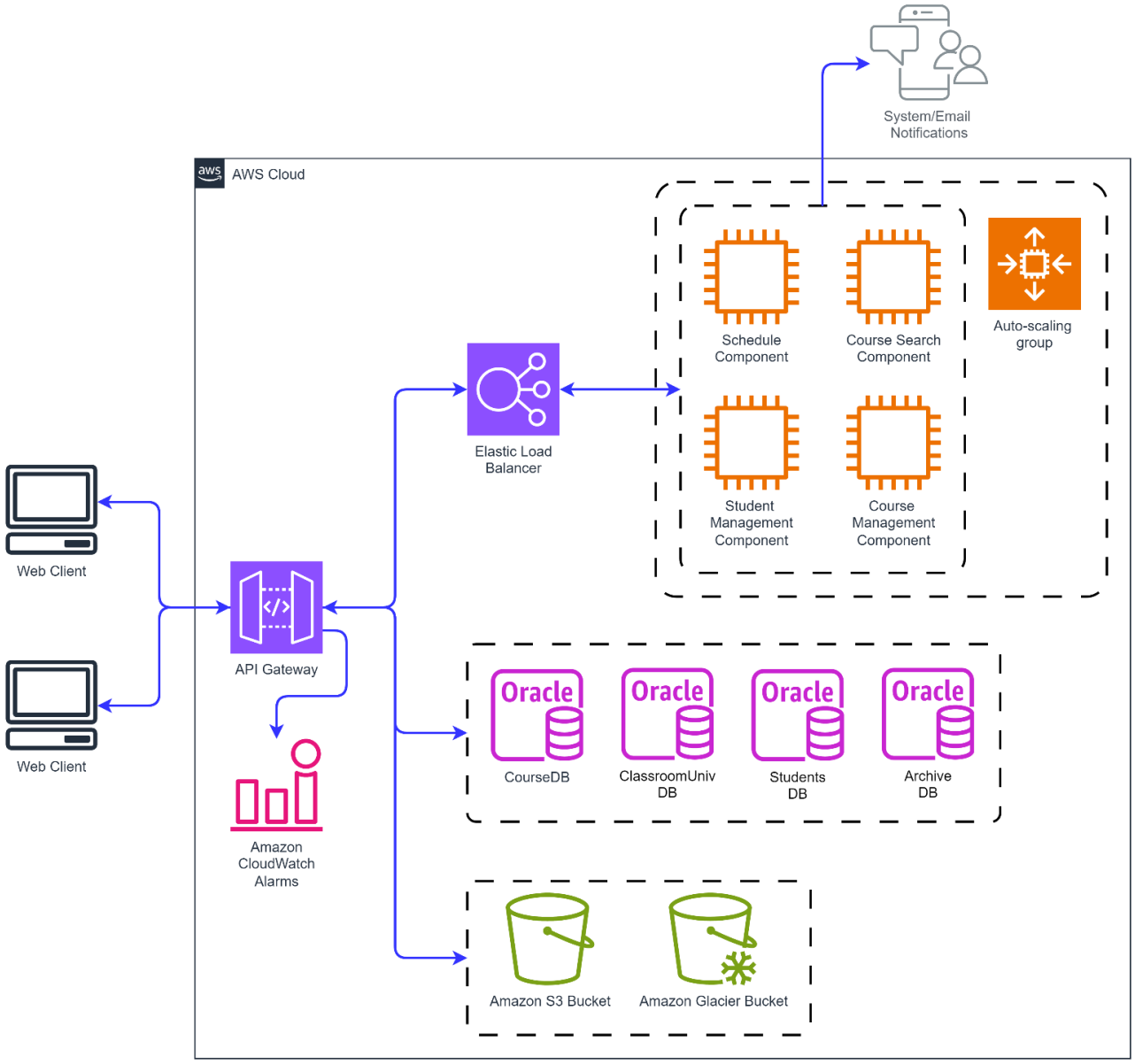
The Course Search component is utilized in nearly all use cases and has data exchanges with all user types. The Course Search component utilizes the Course Database and provides course information to the Schedule component.

The Student Schedule Builder and Professor Schedule Builder UIs utilize the Course Search component and Schedule component to search for courses and build their schedules.

The Course Administration UI utilizes the Course Search component, which provides course information to the Course Management module, e.g., to manage location conflicts. The Course Administration UI also utilizes the Course Management component to add new courses or delete courses. The Course Management component requests and provides classroom data to the ClassroomUniv Database, provides updated course information to the Course Database, and provides deleted course information to the Archive Database.

The Course Administration UI utilizes the Student Management Component to override enrollment information for students and enroll them in courses. The Student Management Component interfaces with the Students Database, which in turn interfaces with the Schedule component for the student's schedule.

Deployment Diagram



Deployment of the Course Scheduling System is pictured here using AWS. An API gateway acts as a single point of access for students, professors, and course administrators to access the system. The API gateway enforces access control policies so that, for example, a student can't access the course management component, but only the schedule and course search components. The API gateway also partners with AmazonCloudWatch alarms to inform university system administrators of issues.

The system utilizes elastic load balancers to manage traffic to instances of virtual machines that contain the scheduling component, course search component, student management component, and course management component. Those components, as part of the system design, send system and/or email notifications to users when changes are made to their schedules.

The system utilizes oracle as its database provider.

The system also uses an S3 bucket for long term storage of data for the classroom, students, and course databases, while the archive database would use the glacier bucket, as its resources would not be accessed as frequently.

Skeleton Classes

```
class Course {
    private int courseId;
    private String courseName;
    private String description;
    private int credits;
    private int numSections;
    private int frequency;
    private int sessionLength;
    private String prerequisites;
    private String school;
    private String startDate;
    private String endDate;

    public CourseInfo[] getCourseInfo(){
        // returns course information (above attributes) as an array
    }

    private void archiveCourse(int courseId){
        // sends course data to the archive database
    }
}

class CourseSection extends Course {
    private int sectionId;
    private int seats;
    // array of days on which the course section takes place
    // array of timeslots in which the course section takes place
    private String professor;

    public SectionInfo[] getSectionInfo(){
        // returns section information (above attributes) as an array
    }

    public void assignClassroom(int sectionId, String school, int seats, int frequency, int
sessionLength){
        // assigns classroom association to course section using logic defined in the
activity diagram
    }

    public void updateProfessor(String professor){
        // changes the value of professor attribute when a professor assigns themselves to a
course section
    }
}
```

Skeleton Tables

Course Database

courseId	courseName	Description	Credits	numSections	Frequency	sessionLength	Prerequisites	School	startDate	endDate	sectionId	Seats	Days	Timeslots	Professor

ClassroomUniv Database

Building	schoolPreference	roomNumber	numberOfSeats	Availability

Students Database

Id	Email	accountPermissions	courseHistory	Major	maxCredits	courseSchedule

Archive Database

archiveDate	archiveUser	courseId	courseName	Description	Credits	numSections	Frequency	sessionLength	Prerequisites	School	startDate	endDate	sectionId	Seats	Days	Timeslots	Professor

Design Patterns

The main design patterns used in this system are the GRASP Principles Expert and High Cohesion.

Assignment of methods to classes was based on Expert, especially as the system was refined. The assignClassroom method was initially part of the Course class, but when considering the fact that some of the inputs and the overall output of the function were specific to the CourseSection class, which extends Course, assignClassroom was moved to the CourseSection class.

In addition, the user classes (User, extended by Student, Professor, and CourseAdministrator) are an example of high cohesion in the system. Each class has its set of characteristics and abilities, and no more than is necessary. The goal is to limit the functionalities afforded to lower-privilege classes like Student and Professor, so that the ability to manipulate course data and override enrollment rules lies only with Course Administrators.

To incorporate cloud-native patterns I chose to utilize an API gateway in the AWS deployment structure. This gateway enforces access control and authorization policies to ensure that each user type is only able to perform the functions of their respective role. The gateway also serves to manage and monitor traffic from system users.

Conclusion

In this project, I was able to outline an object-oriented software system from user identification and requirements definition all the way up to the point at which one would start writing code. This was a valuable exercise for me, as I don't have extensive experience in designing software systems. I feel that this honed useful skills to designing any software system, especially in having the discipline to thoughtfully lay out a system rather than jumping straight into coding.

Next steps for this project would include refining the dependencies between classes to reduce coupling, and to consider adding GoF patterns where they may be of use. I did not focus on adding GoF patterns in this iteration of the project because I didn't want to add patterns for the sake of adding patterns. I did not want to add patterns into the design without having a full understanding of what problems they would solve in the context of the Course Scheduling System.

I also designed the Course Scheduling System in a way that I could hopefully turn this exercise into a full-blown, portfolio-worthy software system.