AC 2009-2507: CPAS: ON THE STRUCTURE AND USABILITY OF A COURSE PLANNING AND AUDIT SYSTEM

Tal Rusak, Cornell University

Tal Rusak is an undergraduate student at the Department of Computer Science at Cornell University, graduating May 2009. Tal's interests lie in understanding the structure of networks and novel computing systems as well as in the theory and practice of engineering and computer science education. Tal was recognized as the 2009 Computing Research Association (CRA) Outstanding Undergraduate Award Winner. Tal's research in modeling the temporal variations of low-power wireless network links has been published internationally and was recognized by the Best Paper Award at ACM MSWiM'08 and two first prizes at ACM Student Research Competitions. In addition, Tal has served as a teaching assistant in varied courses at Cornell, including theoretical computer science and a novel course that introduces the mathematical basis of networks to a diverse body of students.

Christopher Barnes, Cornell University

Chris Barnes is an undergraduate student in the Department of Information Science at Cornell University. He is expecting to complete his BA in the program in May 2009. Chris is primarily interested in the architecture and interface design of web-based applications.

G. Scott Russ, Cornell University

Scott Russ is an undergraduate student in the Department of Information Science at Cornell University. He is expecting to complete his BS in Computer Science in May 2010. Scott is primarily interested in the interface design and product strategy of web-based applications.

Vincent Kam, Cornell University

Vincent Kam is an undergraduate student in the College of Engineering at Cornell University. He is expecting to complete his BS in Computer Science in May 2009. Vincent is one of the founders of the Course Planning and Auditing System, having conceived the idea with two of his friends during his freshman year.

David Gries, Cornell University

David Gries is a professor of computer science and currently associate dean of Computer Science. He received his PhD from the Technische Hocschule Munchen in 1966 and then served on the CS faculty at Stanford for three years. He has been at Cornell since 1969, except for two years at UGA, and served as the Department Chair in the 1980s. Gries is known for his work in compiler construction and programming methodology and his textbooks in compiler writing, programming, and discrete mathematics. He has received several national/international awards for his contributions to education and is a Cornell Weiss Presidential Fellow, awarded for his contributions to undergraduate education.

CPAS: On the Structure and Usability of a Course Planning and Audit System

Abstract

We present CPAS, a system that allows students and their advisors to track progress in an academic program. Our design is able to perform a semi-automatic audit of degree requirements; the only human-approved components are ones that require discretion.

The underlying mechanism is based on robust propositional logic with extensions that make the entry, interpretation, and audit of complex degree requirements straightforward. The system also includes support for human-approved requirements due to the natural discretion that must be applied when evaluating certain requirements, such as multiple courses with a "cohesive theme".

Additionally, we demonstrate an implementation, which focuses on providing a clear, straightforward interface for all users while maintaining the appropriate level of security and privacy. To the best of our knowledge, CPAS is the first degree audit system that was designed with a web-based, graphical interface from the start. CPAS indicates in simple terms how a student is progressing in the degree and allows students to explore other academic programs offered at their university.

We also address several key software engineering research questions, such as the storage of complex major requirements and the design of lucid user interfaces for a variety of users. Considering the number of academic institutions and the variety of academic programs offered, quantifying and collecting correct expressions for major requirements in our system is a nontrivial problem. We explore the possibility of using a collaborative social network, with appropriate security and quality controls, for this purpose. We show how CPAS was used to enter the major requirements of complete academic programs and present a visualization functionality that illustrates such programs.

CPAS is a fundamental contribution to education research since it provides a way for academic programs to be mapped out in a generalized ontology. Thus, it allows students to maximally utilize the academic resources of their university, and it allows faculty members and departments to plan and represent programs and to advise students effectively.

1 Introduction

Traditionally, course selection and degree audits at Cornell and many other higher education institutions have been performed with paper and pencil tools or individually designed spreadsheets used by both students and departments. A fully automated solution is impossible, since some requirements need approval of the advisor or involve vague guidelines only, for example, that two courses must be "in related disciplines". Our Course Planning and Audit System (CPAS) integrates automatically auditable requirements with those that must be manually approved in a straightforward way using a simple, well-designed web-based interface. CPAS allows degree requirements to be easily specified by department staff in a very general way. It was used in a prototype system to define the computer science degree requirements in the Engineering and Arts & Sciences colleges at Cornell and several majors at other universities.

CPAS was structured using well-established methodologies from the theory and practice of software engineering and computer science. We use simple logical foundations for the back-end of the system, where the major components of a degree auditing system—courses, students, and requirement descriptions—are stored. This leads to a simple and robust representation of the major requirements and users of the system. At the front-end, we draw from principles of web design and user interface research in order to provide an easy-to-use, generalized system to support a wide range of disciplines and users of all experience levels and roles.

The rest of the paper is organized as follows. Sec. 2 provides background, definitions, and presents related work. Sec. 3 defines user roles in the system and describes the security model for CPAS. Sec. 4 gives an overview of the logic that underlies our system and its extensible object-oriented implementation. Sec. 5 discusses user interface considerations and provides an overview of the usability of the system. Sec. 6 provides an overview of the applications of CPAS to studying academic programs using the proposed formal representation. Sec. 7 presents possible architectures for the system when it is deployed. Finally, Sec. 8 presents discussion and conclusions.

2 Background and Related Work

A bachelor's degree is granted when a student completes the required coursework assigned by the faculty. There are two fundamentally different types of requirements in higher education: *automatically auditable* requirements and *manually auditable* requirements. Examples of the first type are a single course taken by all students (e.g. CS 101) and a requirement expressed by simple propositional logic (e.g. $(\mathsf{CS\ 282} \wedge \mathsf{CS\ 283}) \vee \mathsf{CS\ 284}$). As we show in Sec. 4, some automatically auditable requirements cannot be expressed in such simple terms.

Manually auditable requirements, by their nature, *require* human intervention to be approved. Typical examples include "Advisor approved electives" or loosely defined requirements such as "two courses in a related field". In such cases, deciding whether a student has fulfilled the requirement is up to interpretation and approval. Under certain circumstances, there may be requirements that transcend these categories, for example, that a student must either take CS 132 or take another course approved by the advisor.

Several automated degree audit systems exist, and some are widely used. For example, the PeopleSoft Academic Advisement Module features a complex database-table entry for academic program requirements.[8] Other similar systems include the u.achieve system (formerly DARS, Degree Audit Reporting System) from Miami University[2;9] and the Degree Navigator Plus from Decision Academic Plus.[1]

CPAS contributes a simple grounding in logic that allows major requirements to be represented as trees, a basic concept for data representation that has its roots in computer science. In addition, we show an easy-to-use, web-based user interface for students, staff, and faculty advisors. We also ensure that information is clearly and concisely represented and suggest a tree-based visualization scheme that can represent major requirements generally. We defer a more detailed comparison of a key aspect of the user interface with the design of previous systems to Sec. 5.2 and a discussion of our visualization scheme to Sec. 6.

Permissions for Various Users of CPAS

Users	*Role*
Student	Record/Update courses, track their own progress in the degree
Faculty Advisor	View the progress of their advisees, approve certain types of manually auditable requirements
Staff	Add/Edit/Delete courses, Add/Edit/Delete major requirements within their department, view progress of students in their department, Create/Edit faculty advisor and student accounts
Supervisor	(All staff user permissions) + Create/Edit accounts, Add/Edit/Delete majors

Table 1: CPAS user account roles and associated permissions.

3 CPAS User Roles

CPAS user roles are based on the principle of least privilege—each user gets "the least amount of privilege necessary to complete the job".[3;10] There are two user types: student and administrator. The administrative category is further divided into three types, with increasing permissions: faculty advisor, staff, and supervisor. Table 1 gives an overview of the activities that each user can perform.

4 Course/Requirements Logic

4.1 Coursework Logic

The underlying logic for requirements starts from appropriately categorizing all the courses. Courses are naturally organized by department and assigned a number, and CPAS allows each course to be labeled in this way. However, courses can be related in ways that transcend traditional departments. At Cornell, for example, a number of courses from disciplines as varied as computer science and industrial and labor relations are considered "information science" courses. Similarly, courses from a variety of departments fall under the "Cultural Analysis" group that can be used to help satisfy liberal studies requirements for Cornell Engineering students. Such groupings are not unique to Cornell. Many universities have started interdisciplinary programs, such as information science or symbolic systems programs, which relate courses in a variety of areas.

Official and unofficial course groupings are often used when writing requirements for academic programs. To account for this observation, CPAS allows the creation of an arbitrary number of such groups, called *categories*. Staff can assign any number of courses to each category, or the information can be parsed out of appropriately-formatted documents. CPAS records courses taken by students, keeps track of grades, credit hours, and semester and year taken, and it is extensible to support student information systems.

4.2 Requirements Definitions

Next, we consider the maintenance and tracking of requirements. A strength of CPAS is the simplicity, consistency, and generality with which it maintains academic requirements. CPAS uses three types of requirements to represent the complex nature of advanced academic programs: *individual courses*, *option groups*, and *free-text requirements*.

Individual courses are specific courses, represented by the department name and course number, offered by the university. Individual courses can also have attributes such as whether they can be taken pass/fail, a minimum grade required, and the minimum number of credits necessary.

Option groups allow more flexibility than the individual course requirement. Each option group has one set of top-level rules and an arbitrary number of other rules. The *top-level rule* specifies the number of courses to be taken and the overall number of credits that need to be achieved (e.g. 6 courses totaling at least 18 credits). *Course-level rules* specify that a certain number of those taken to satisfy the group must be above, at, or below a certain level, as specified by the university's numbering scheme (e.g. 3 courses must be numbered at least 2000). *Department rules* specify that a certain number of courses must or must not be in a certain department (e.g. courses may not be in the Computer Science Department). *Category rules* specify that certain courses must or must not be in specific categories (e.g. 6 courses must be liberal studies courses), and *category option rules* specify that courses representing one or more of the categories must be taken (e.g. a course must be taken from at least 3 of the following 6 categories). Finally, *specific course rules* allow or disallow the use of certain courses to fulfill the requirement (e.g. EDUC 2400 is allowed or is not allowed). Figure 4(d) shows the interface used to define an Option Group. The Option Group was inspired by the Liberal Studies requirement for Cornell Engineers. In a nutshell, the liberal studies requirement consists of at least 6 courses for a total of at least 18 credits, two courses of which must be at the 2000 level or higher, and at least one course in three of six categories (e.g. language, performance, history, and culture). As shown below, this construct was useful in several other places in designing the requirements for a number of other majors that we represented using the system.

While full academic programs can be represented with the aforementioned logical expressions, we found that this presents two challenges. The first is the extreme complexity of the resulting expression, which would be difficult to input correctly. Secondly, requirements change over time and are usually associated with graduation class years. In many cases, this change is evolutionary, i.e. most parts of the program remain the same but certain parts are updated.

We found that it is better to separate the overall expression into individual *requirement units*, all of which must be fulfilled. A requirement unit can be loosely defined as a high level requirement covering a certain theme, e.g. "Liberal studies", "Scientific Computing", "Operating Systems", and "Calculus sequence". The division of the degree requirements into these units is up to the user of the system but is most likely quite obvious from the way requirements are specified in the university catalog. Since requirements are likely to evolve in terms of these units, CPAS associates certain class years with each unit and dynamically builds the expression corresponding to units belonging to each class year. Since variations are often minor, CPAS makes it easy to clone requirement units when starting to construct new units for subsequent class years.

At many institutions, students must first affiliate with an academic field and then complete the full academic program for their chosen discipline. Thus, CPAS maintains two requirements expressions for each class year, one representing progress toward affiliation and the other toward graduation. In addition, some institutions have program requirements that change the curriculum but do not require a unique course to satisfy. For example, a student may be required to take a course in probability theory, regardless of whether it is a free elective, a core elective, a liberal studies course, or an engineering distribution course. We call such requirements *structural requirements*. CPAS distinguishes each of these requirements with a type label: affiliation-only, graduate-only, affiliation and graduation, and structural (non-course) requirement.

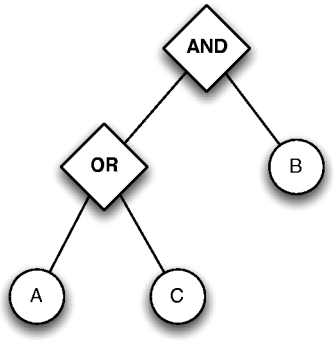


Figure 1: Example of requirements unit tree. Diamonds represent logical operations and circles represent major requirements.

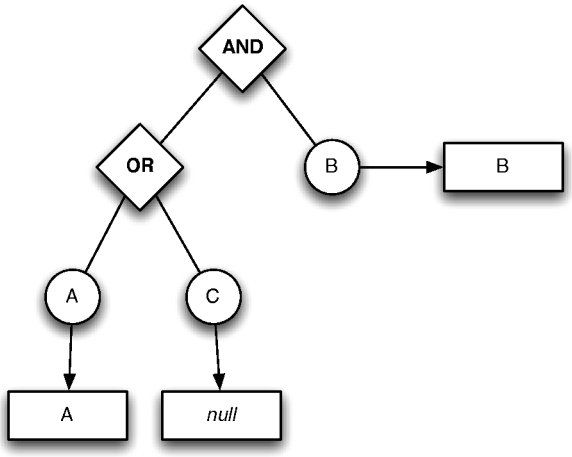


Figure 2: The requirements unit defined in Figure 1 mapped to courses that fulfill these requirements. Diamonds represent logical operations, circles represent major requirements, and rectangles represent courses applied to satisfy the requirements. A rectangle labeled "null" means that no course taken has satisfied the corresponding requirement.

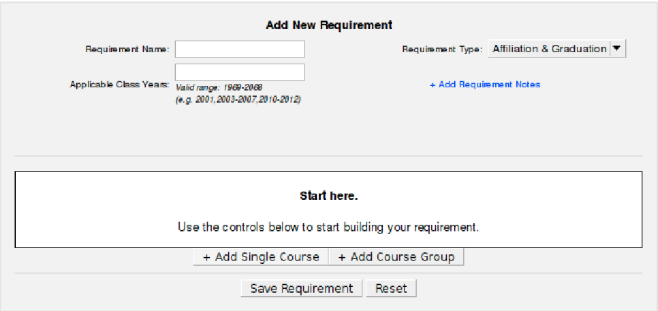


Figure 3: CPAS requirements unit interface. The header information (name, class years, and type) is common to all CPAS requirements. Note the clear starting point in building a requirement.

4.3 Requirements Audits

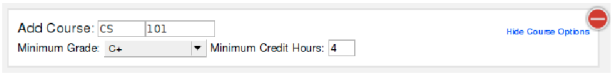
Graduation and affiliation audits are straightforward to perform using well-known computer science techniques. The requirement units form a "binary tree" representation of the logical expression, and it is easily evaluated. By maintaining a database of such expressions for each student, audits can be performed by evaluating the validity of each node in the tree from the bottom up. If the root of the tree evaluates to TRUE, then the affiliation or graduation requirements are completed. On the other hand, any requirements unit that evaluates to FALSE is flagged in the user interface to indicate to the student which requirements remain to be completed if the node does not evaluate to TRUE.

Figure 1 illustrates a sample tree that may be used to represent a requirements unit. Assume that the student is required to take course B and either course A or course C. The student has taken courses A and B. Figure 2 shows how these courses may be placed in the requirements tree. Node A is TRUE, node C is FALSE, the OR node evaluates to TRUE, node B is TRUE, and the AND node evaluates to TRUE. Since the AND node is the root of the requirement, the entire requirement evaluates to true. The student has fulfilled this requirements unit.

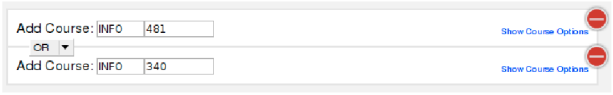
5 User Interface and Usability

The CPAS interface takes into account each of the user roles discussed in Sec. 3 and the logical grounding considered in Sec. 4. In designing this interface, we adhere to the principle of information hiding and abstract[7] as many details as possible from the user. Unlike many previous degree audit systems, CPAS was designed with a graphical user interface from the start. The interface is the result of extensive design and user testing. The basic concepts followed in implementing this design are covered by Gould and Lewis.[6] CPAS has a universal inline help system, which provides assistance to users on every page of the system, and pervasive search and filtering features for information such as users or courses.

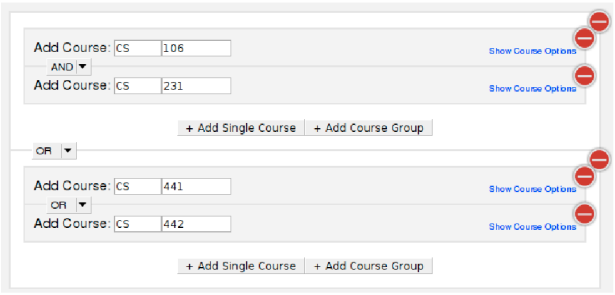
The user interface is designed to be accessible to all users—most elements are basic XHTML components with an underlying CSS style sheet and JavaScript code for dynamic components. Most components are scalable and can be increased in size or used with screen readers to aid in accessibility. Furthermore, color is never the only way used to convey information.[5]



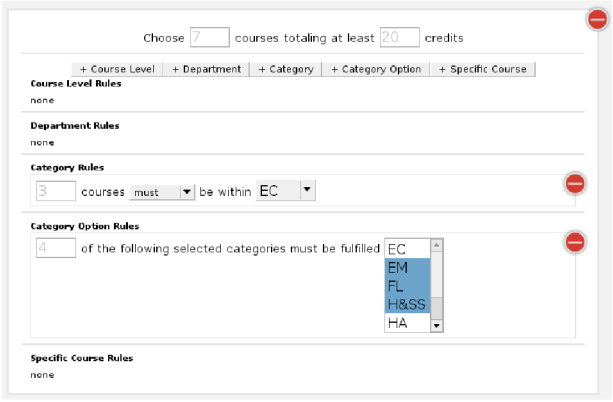
(a) This screen demonstrates the entry of a single course and optional elements (grade and credit hours)



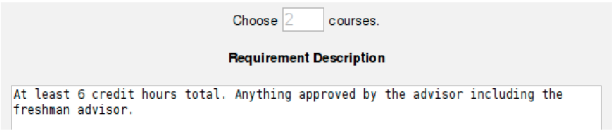
(b) This screen shows the use of Boolean expressions



(c) This screen shows the application of course groups to achieve parenthetical elements in the requirements expressions.



(d) This screen shows an option group used to input complex requirements



(e) This screen shows the entry of a manual requirement.

Figure 4: Examples of Requirements Entry in CPAS.

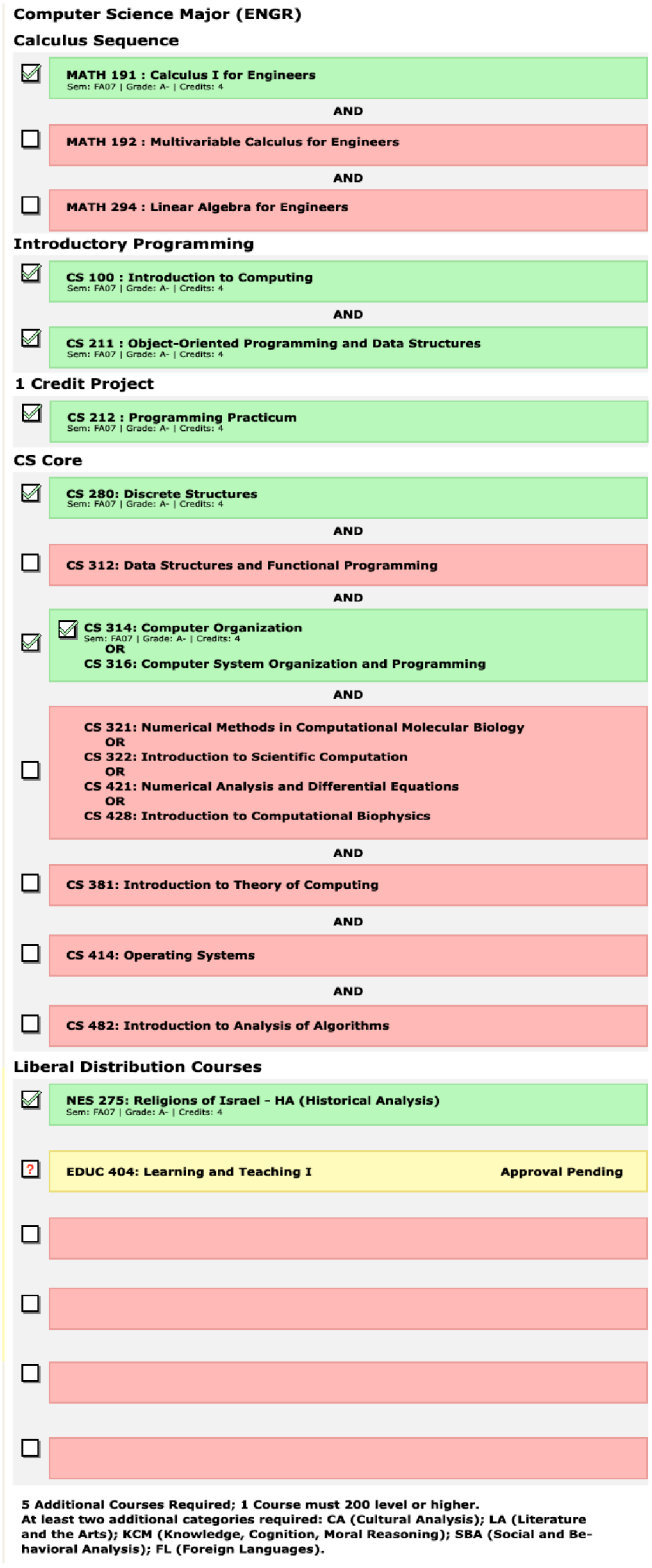


Figure 5: Requirements tracking visualization. The purpose of this screen is to let the user know, in a quick glance, how far a student has progressed in completing the degree.

5.1 Requirements Entry Interface

Figure 3, illustrates the initial interface displayed to staff members when creating a new requirement unit. Note the presence of requirement class year(s), requirement type (affiliation only, affiliation and graduation, or graduation only), and the ability to add free text notes to any requirements. In initial versions of this interface, users were confused about where to start building the requirement, so we added a clear starting point as shown in Figure 3.

Figure 4 shows the interface used by staff members utilizing the system to enter requirements into each requirements unit. Figure 4(a) is an individual course that is used for a requirement. As shown, it is possible to optionally enter a requisite grade or number of credit hours required for each course. If these options are not specified, then CPAS defaults to a global setting that can be specified per university policy. In Figure 4(b), we see several courses separated by Boolean operators. In Figure 4(c), we demonstrate the use of groups, which simulate parenthetical elements in the logical expressions. Figure 4(d) demonstrates the capabilities of the option group, and Figure 4(e) shows the entry interface for a manually auditable requirement.

5.2 Requirements Tracking

The display of progress made toward fulfilling the degree requirements is of central importance to a degree audit system. Previous systems have tried varied approaches. The DARS system (now u.achieve), first developed in the 1980's, originally used text based output in a specific format to convey this information and was gradually improved to include color codes and then graphical output. The current version displays a number of graphs as well as a list of courses. Different information is displayed as the mouse pointer hovers over each. This version does not completely omit the original text-based formatting in the list of requirements.[2;9] Original versions of Degree Navigator illustrated different degree requirements as graphical "islands" on a map.[1;9]

Unlike the previous approaches, CPAS indicates in simple terms how a student is progressing in the degree. Each requirement is represented by one or more boxes, illustrating the courses or paths that can be taken to fulfill the requirement. Many courses can be applied automatically. In other cases, however, students can choose how to apply their coursework toward a requirement. For example, a course might be an advisor approved elective. In such cases, CPAS allows students to choose how each course applies.

The CPAS interface for authorized users to track student progress is presented in Figure 5. Requirements that are not fulfilled are red and have an empty check box next to their box. Requirements to which courses have been applied are either approved immediately (if automatically auditable) or sent to appropriate staff for approval if they are manually auditable. Courses that are in the process of being approved are yellow and a question mark is displayed in the check box. Finally, once any course is approved it becomes green and the check box is filled in. Note that the check boxes along with the written comments have an obvious meaning even without referring to the color, increasing accessibility to colorblind people.[5]

6 Visualization and Representation of Academic Programs

In addition to the clear role of the CPAS as a tool for students, faculty members, and department staff persons, we can also use the ontology provided by the logical representation in CPAS to understand academic programs. We have demonstrated the mechanics of CPAS and the logic

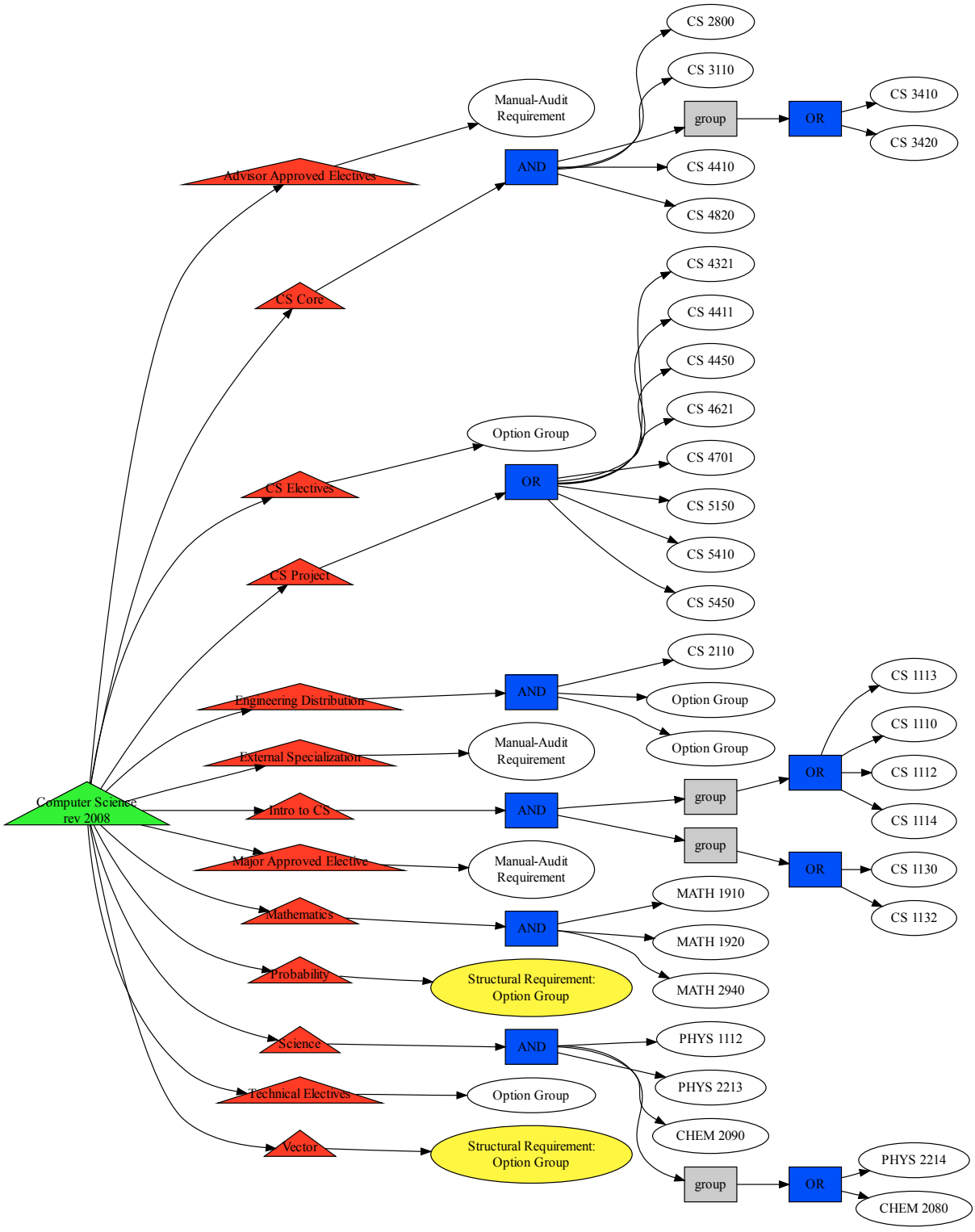


Figure 6: A tree-based representation of the new requirements in Computer Science at Cornell, introduced in Dec. 2008.

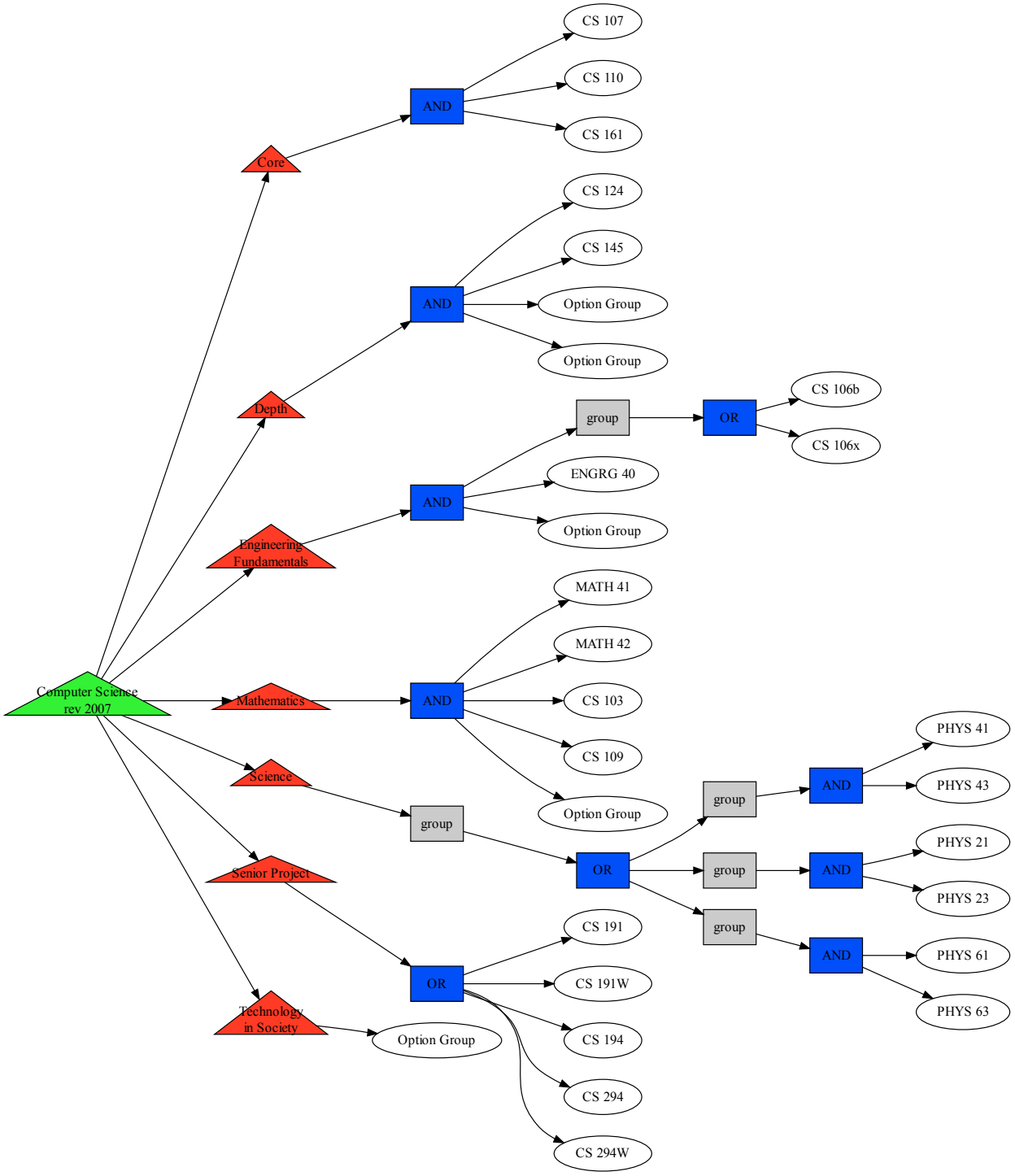


Figure 7: A tree-based representation of the requirements for the new Computer Science major (Information Track) at Stanford, introduced in 2007.

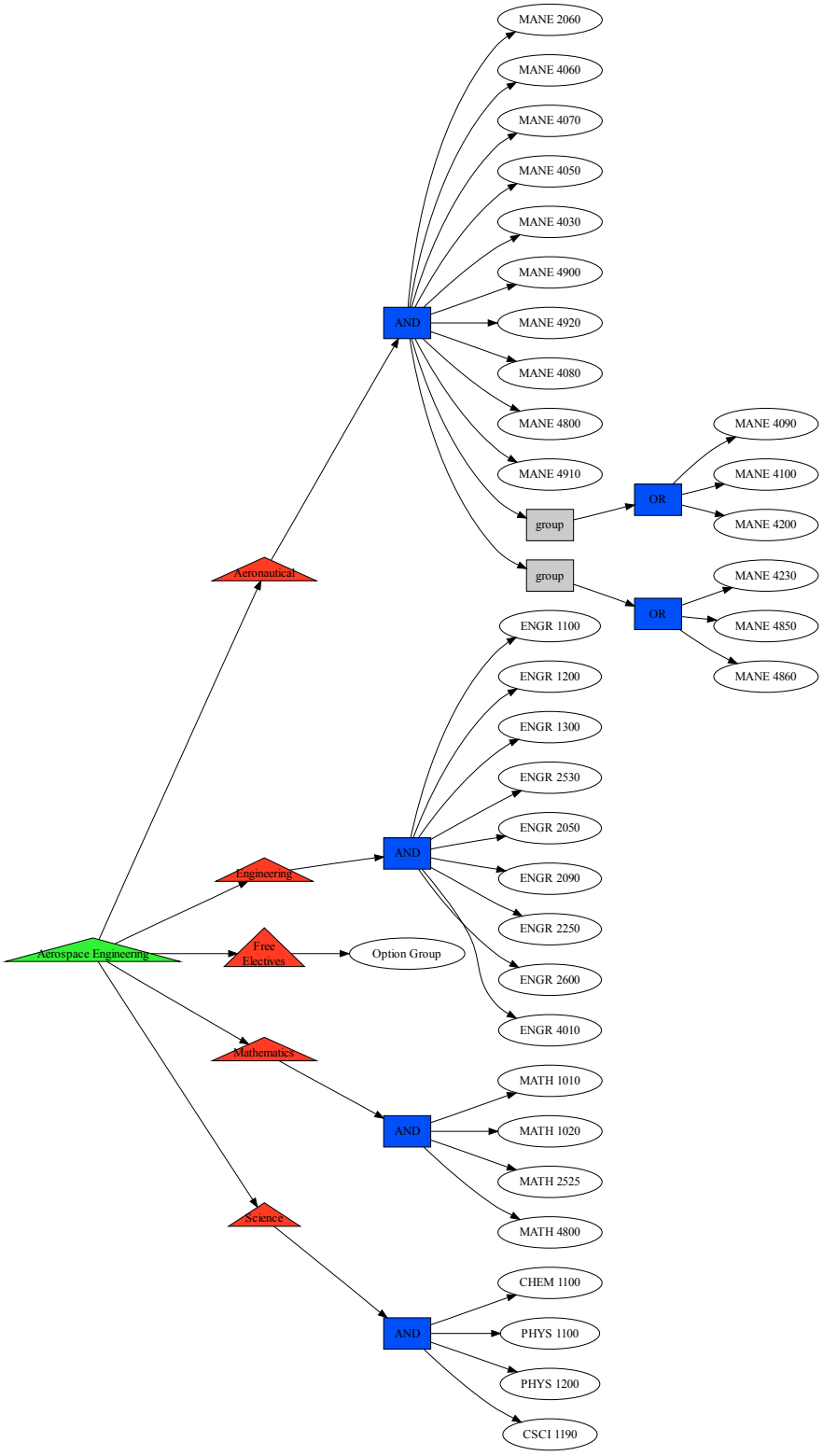


Figure 8: A tree-based representation of the requirements for the Aerospace Engineering major at the Rensselaer Polytechnic Institute.

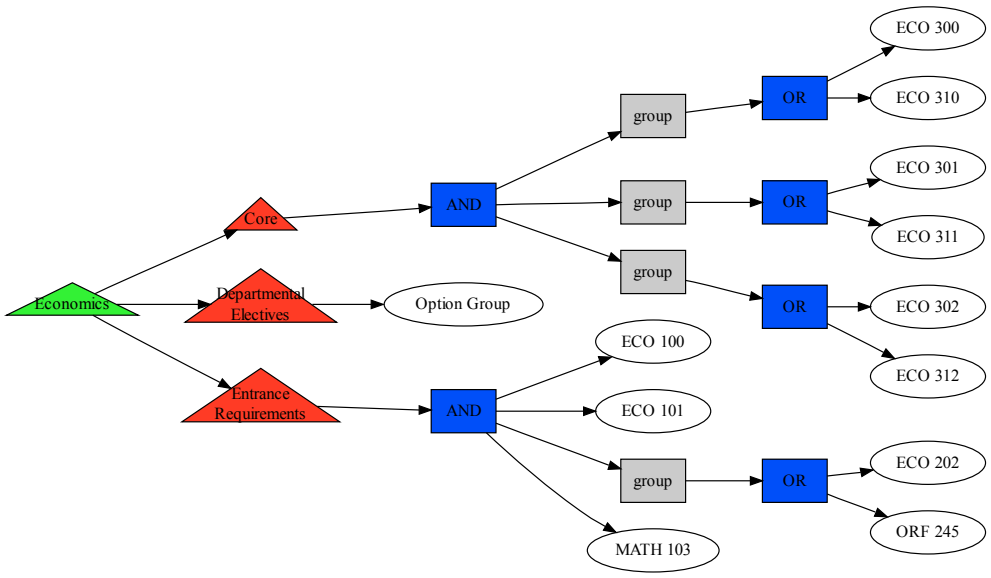


Figure 9: A tree-based representation of the requirements for the Economics major at Princeton.

behind our system; now we demonstrate its broad applicability to a wide range of university and curricula. We also consider the system's possible usage schemes to represent a wide variety of programs. Then, we focus on the option group and show how it can be used to express a range of liberal studies programs in various Engineering programs.

In contrast to the widely varying formats in which universities publish this data, our work provides a unified scheme by which to represent these requirements. This may lead to a better understanding of a curriculum for a major.

6.1 Departmental and College Requirements

To show the generality of our system, we show the tree-based structure discussed above in the representation of actual academic programs from a wide range of universities and majors for college and department requirements. We defer a discussion of core requirements and liberal studies requirements for engineering to the next subsection. The trees, constructed using the GraphViz package,[4] aim to show the underlying structure of each major tree. Such trees may be used in the further analysis of academic programs for discovery of errors, omissions, or similarity among the academic programs at different universities. We are interested in pursuing this as future work.

Figure 6 shows the requirements for Computer Science at Cornell University using the tree format, while Figure 7 shows the Computer Science major at Stanford University. Figure 8 shows the Aeronautical Engineering major at the Rensselaer Polytechnic Institute. To show an example of a non-engineering major expressed with our system, we present Princeton's Economics major in Figure 9.

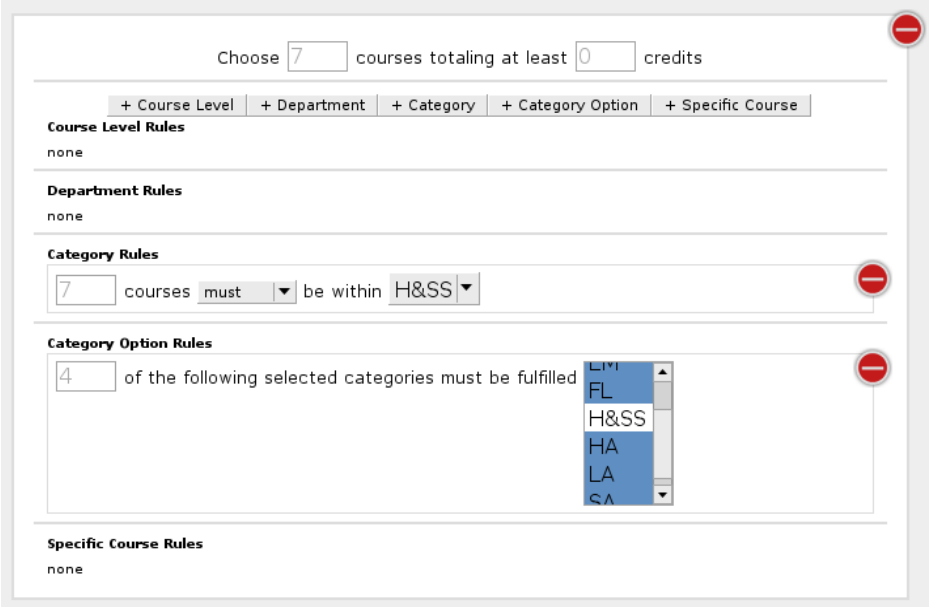


Figure 10: Distribution Requirements for Engineering Majors at Princeton.

We can already come to preliminary conclusions about these majors. The RPI Aerospace Engineering major and Princeton's Economics major have highly structured requirements. Similarly, we can conclude that Cornell's Computer Science major has a more complex characteristic, including two structural requirements, than that of Stanford.

6.2 Liberal Studies and Core Curriculum Representations

In this section we show examples of liberal studies requirements that can be represented in our system's ontology. We express a wide variety of liberal studies, general education, and other non-engineering courses that are required by engineering students at a variety of universities. Figure 10 shows a single option group representing Princeton's Distribution Requirements, Figure 11 shows several option groups representing General University Requirements for Engineering Majors at the New Jersey Institute of Technology. Finally, Figure 12 shows the Liberal Studies requirement at Cornell University as a single option group. By expressing these examples, we demonstrate the generality of the option group for the representation of liberal studies courses for engineering students at a wide variety of universities.

7 Proposed Architectures for CPAS

We considered two overall architectures for CPAS. In the first model, CPAS is fully supported by a university, which integrates the system into its support infrastructure for students, trains staff members, and arranges for the entry of many major requirement trees into the system. In the second model, CPAS is implemented as a social network, where major trees are input and are maintained by the community. We applied the results of user studies in the analysis of each of these architectures.

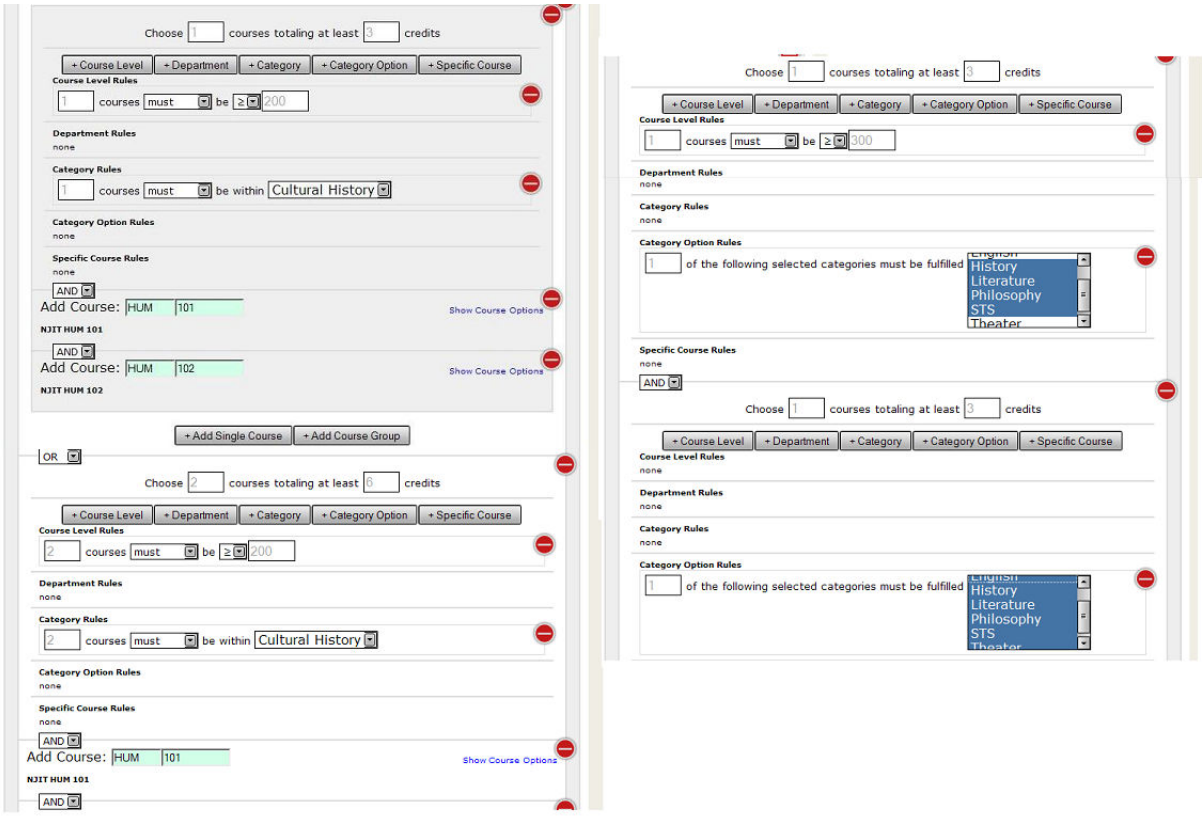


Figure 11: General University Requirements for Engineering Majors at the New Jersey Institute of Technology.

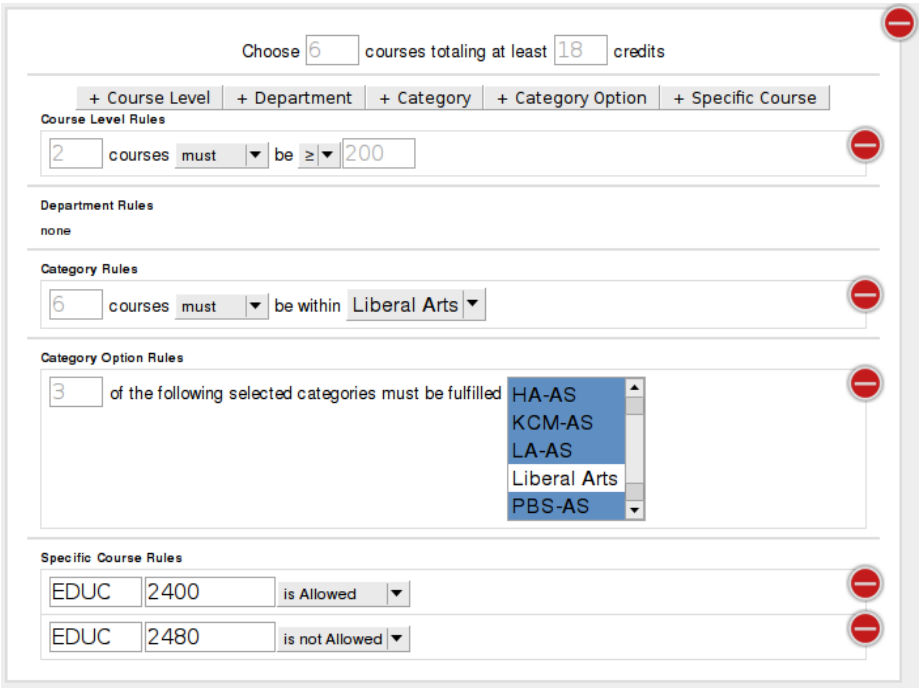


Figure 12: Liberal Studies Requirements for Engineering Majors at Cornell.

7.1 University-Centric Model

In this model, CPAS would be given or licensed to universities for the purposes of auditing students' academic records as an administrative tool. In such cases, departmental staff would be responsible for entering the requirements of each major and ensuring that the major is correctly represented. In turn, students would enter their coursework, or such coursework would be imported automatically from university information systems such as PeopleSoft.

The major technical challenge in this approach is the standardization and interfaces between complex systems that need to interact, especially if student coursework is entered automatically. We continue to pursue this model concurrently with the social networking model discussed below.

7.2 Social Network Architecture

In the absence of the support of an institution, we envision CPAS to be used as a tool where a collaborative community could create major requirements and a large number of users could audit and plan their academic careers using the major requirements already entered into the system. In turn, such users would correct errors and make additions to missing or incomplete majors. In essence, this model applies the principle of Wikipedia to CPAS.

There are several possible issues with this model. First, entering major requirements is difficult and takes time and effort to do correctly. It is not clear that anyone would have the incentive to correctly enter major requirements for others' benefit. It is also not clear how appropriate quality controls can be put in place to ensure that the majors are all entered correctly. One possible way to partially remedy these issues is to allow for a system of voting for the quality of representations and the assignment of trusted editors—either the system's maintainers or university staff members who volunteer—to check the representation of the majors. These are areas that we are actively investigating and that we will cover in future work.

Thus, we believe that this approach may be workable if there is a solid system outlining the participation guidelines and quality controls. In addition, it would likely be necessary to add a core of majors or to provide incentives to users to add their majors into the social network.

8 Discussion and Conclusions

CPAS combines a simple, robust logical framework with an elegant, simple user interface and visualization in order to help students, advisors, and staff track degree requirements. Majors can be represented concisely and with a common ontology, thereby allowing for a general visualization of major requirements across many disciplines.

The immediate contributions of CPAS allow students to efficiently track their coursework and to understand which requirements they have completed, allow advisors to suggest courses to students and track their students' progress, and allow departments to visualize their academic program in a standardized way. Interesting open questions are how such a system can be used to compare major requirements of various universities and/or to find improvements or errors in existing academic programs. By presenting a standardized format in which major requirements can be represented, this work provides a way for these questions to be answered more effectively.

Acknowledgments

We appreciate the role of Patrick Coffey, Keith Bodin, and Sarah Perkins in the design and implementation of portions of CPAS. This work was supported by a generous gift from Cisco Research. Extensive discussions with Sara Lin, William Arms, and Carla Gomes helped in developing and improving our ideas. We also thank Cornell's Computer Science Department for its support of CPAS and especially appreciate the work of Nicole Roy, Gloria Loehle, and Dora Abdullah.

References

1. Decision Academic Inc. Degree Navigator Plus, `http://www.decisionacademic.com/`, 2009.
2. redLantern u.achieve, `http://www.redlanternu.com/uachieve/`, 2009.
3. C. Botev, H. Chao, T. Chao, Y. Cheng, R. Doyle, S. Grankin, J. Guarino, S. Guha, P.-C. Lee, D. Perry, C. Re, I. Rifkin, T. Yuan, D. Abdullah, K. Carpenter, D. Gries, D. Kozen, A. Myers, D. Schwartz, and J. Shanmugasundaram. Supporting workflow in a course management system. *SIGCSE Bull.*, 37(1):262–266, 2005.
4. J. Ellson, E. Gansner, L. Koutsofios, S. North, and G. Woodhull. Graphviz–Open Source Graph Drawing Tools. *Lecture Notes in Computer Science*, 2265:483–484, 2002.
5. B. Friedman. Value-sensitive design. *Interactions*, 3(6):16–23, 1996.
6. J. Gould and C. Lewis. Designing for usability: key principles and what designers think. *Communications of the ACM*, 28(3):300–311, 1985.
7. K. R. M. Leino and G. Nelson. Data abstraction and information hiding. *ACM Trans. Program. Lang. Syst.*, 24 (5):491–553, 2002.
8. Oracle Inc. *PeopleSoft Enterprise Academic Advisement 9.0 PeopleBook*. 2006.
9. D. Peters. A Practical Application of Cognitive Work Analysis: Transforming a Static Report Into an Interactive Interface. Master's thesis, Miami University, 2005.
10. J. Saltzer. Protection and the control of information sharing in multics. *Communications of the ACM*, 17(7): 388–402, 1974.