Liberating Legacy System Data With Rails

Stuart Wolthuis
stuartlw@byuh.edu
Brigham Young University – Hawaii
Laie, HI

Christopher Slade
christopher.slade@byuh.edu
Brigham Young University
Provo UT

Abstract

In this research project we describe the legacy software landscape, its current state, and challenges associated with aging information systems and access to its data. We briefly describe the popularity of dynamic languages and how a specific dynamic programming language, Ruby on Rails (RoR or Rails), is used to create a system to extract data from a legacy system to increase efficiency and productivity in an academic class scheduling system. As an example, we describe, first, how a system developed in Rails, called Class Scheduler, pulls data from a legacy student management system (MAPPER) developed in Tcl (pronounced "tickle") and uses this data to vastly increase the efficiency of the scheduling process and, second, how it reduces conflicts in class schedules. We discuss the advantages of automatically extracting and processing the data from the legacy system and the limitations associated with this process.

Keywords: Ruby on Rails; RoR; Rails; legacy systems; legacy data; software engineering; programming; scheduler; class scheduling;

1. INTRODUCTION

With the advent of the cloud and the use of Software as a Service (SaaS) an enormous amount of data produced and stored in legacy information systems can be left behind and left inaccessible unless solutions to extract this data are realized. Without modernization users can keep using legacy systems as they exist and hope the hardware and operating systems providing access to their valuable data continue to function. As a benchmark to the state of large information systems a 2016 Government Accountability Office (GAO) report to Congress stated that in 2015 $61.2 billion was spent on operations and maintenance of current (legacy) systems while $19.2 billion was spent on development, modernization, and enhancement (Powner 2016). Another insightful indicator reported by the GAO is that the amount of IT spending on development, modernization, and enhancement from 2010 to 2017 declined by $7.3 billion, a 28% reduction. This implies that enhanced digitization, which may equate to access of legacy data, is not a current priority for these maturing systems due, in part, to its very costly nature given the three imperatives of data migration: don't interrupt current business processes, maintain data consistency, and effort and cost should be minimized (Martens, Book, Gruhn, 2018).

Aging government information systems include two master files (individual tax files and business tax files) in the Department of the Treasury that are approximately 56 years old with no specific plans for updates (Powner, 2016). Also cited in this report is the fact that the nuclear command and control system is 53 years old and still runs

on an IBM Series/1 Computer with an 8-inch floppy drive.

Systems that manage inmates in prisons, including their security, custody levels, and work assignments is over 35 years old and your SSN is managed by a system that is 31 years old (Powner, 2016).

Additionally, the 12 government agencies in this report indicated using unsupported operating systems; 5 reported using 1980s and 1990s Microsoft operating systems, with no support from the vendor for over a decade.

Any organization that continues to use antiquated technology systems must pay a premium for staff or contractors with the right knowledge to support and maintain legacy systems (Powner, 2016). For example, the author of this paper personally managed the software development of a product improvement program in 1993 with a defense contractor that required pulling an employee out of retirement to change 160 lines of code in a complex system originally written in Fortran 66.

A final note on government legacy systems: the Department of Commerce runs a system providing warnings to the public and emergency managers with several obsolete operating systems: Windows Server 2003 no longer supported by the vendor and a version of Oracle no longer fully supported by the vendor (Powner, 2016). These systems observe meteorological incidents that could generate a tsunami or hurricane.

Both the data contained in legacy systems and the systems themselves require expertise and innovation to maintain their integrity. In this paper we will specifically address the challenge of accessing and using legacy data.

1.1 Purpose of this Research

Demonstrating a middleware solution to access and use legacy data is the purpose of this research. Middleware, including the API and wrapper, became a popular solution to unlock business value (Thiran, Risch, Costilla, Henrard, Kabisch, Petrini, Hainaut, 2005) by exhuming legacy data from aging and sometimes antiquated systems. Persistence (Thomas, 2008) has also emerged as a viable tool in the hands of programmers who need to unearth data secrets that otherwise would remain buried with maturing software. Users require and expect access to mountains of data right now; this is partially driving the need to reach into legacy systems and provide insight via the smart device in the palm of their hand.

All three of these solutions (API, wrapper, and persistence) find their genesis in dynamic programming languages, but come at a cost with additional runtime checking required (Paulson, 2007) since more instructions must be evaluated at runtime, a fact that is probably moot with the realization of new computing platforms (cloud and SaaS) made possible with Next Generation IT (Thomas, 2008).

The Tiobe Index (Paulson 2007) indicates a significant rise in the use of dynamic languages at the time of the referenced report; further comparison to the June 2019 Tiobe Index (Tiobe, 2019) indicates the use of dynamic languages is still 50% of the top 20 most popular languages in the world, with Python, the most popular dynamic programming language, showing "an all time high in the Tiobe index of 8.5%" with more growth expected in the future. This is significant since dynamic languages are used to create middleware needed to extract legacy data from older information systems.

The other nine dynamic languages in the top 20 June 2019 Tiobe Index report have a sum total popularity of 14.8%. In the middle of this list of 9 is Ruby on Rails with a score of 1.388%.

2. Using Ruby on Rails for Middleware

As a platform to create middleware Ruby on Rails (RoR) is distinctively suited with a framework following the Model View Controller (MVC) design pattern (Scharlau, 2007). By definition, Rails is a framework built on Ruby, allowing programmers to develop database-focused websites with scaffolding and code generation (Meenakshi, 2015).

In addition to use of the MVC architecture, RoR uses the Create, Read, Update, Delete routing engine to interact with web pages and follows the concept of DRY: "Don't repeat yourself" (Meenakshi, 2015). RoR is taught at many universities as an upper level class to teach skills required to build dynamic websites as part of the computer science curriculum. Specifically, we will explain how it can also be used to build middleware and a system to improve scheduling of classes in a complex environment.

2.1 The Environment

Our private university hosts about 3,000 students, half from international locations, who pursue bachelorette degrees in the sciences, arts and letters, and professional programs such as business and accounting. Specifically, the authors offer majors in computer science, information technology, and information systems.

Over 20 years ago, a colleague, who is now retired, created an online student management system in Tcl (pronounced Tickle) that allowed academic advisors, faculty, and students to manage and plan a student's academic program. This system mapped a student's classes to complete a major, general education, and minors and is named MAPPER. This ability to "map" or plan the future is especially valuable and sets MAPPER apart from the ERP system (PeopleSoft) used by the registrar, which does not "map" the student's future classes.

Additionally, MAPPER allows academic advisors to document appeals, notes, and guidance to students. Grades are also recorded in this system and transfer credits documented.

The development and feature improvement of MAPPER occurred over decades and was continual, based on input from users, primarily student advisors and faculty. Ironically, other systems used to schedule classes did not improve; spreadsheets are still the norm among many academic faculties, departments, and colleges. This may be evident in the fact that 22% of universities practice "just in time" (JIT) scheduling, planning their next term only one academic term in advance (Hanover, 2018).

3. Scheduling Systems

Research studies show that scheduling is one of the most important and demanding factors impacting student retention at universities (Hanover, 2018). With imperfect tools classes can inadvertently get scheduled at times that interfere with core classes or additional required classes, such as labs. As curriculums and class offerings become more varied and complex the likelihood of conflicts increase. Add to that limited classroom space and multi-use, or specialty (cyber-security sandbox lab, science labs) or high-demand classrooms the scheduling challenge becomes a multifarious problem.

A review of several scheduling systems, including UniTime and Mimosa Scheduling Software revealed very capable systems (Ngoc, 2015) but they did not have the ability to import conflict matrix data from MAPPER. Therefore, a custom development was necessary.

3.1 Scheduling System Challenges

The Higher Education Scheduling Index (HESI) annual report of 157 institutions, including four year private, four year public, and community colleges discovered that classroom utilization is 67% and seat utilization is 62% even though institutions expressed they felt they were out of space (Ad Astra, 2016). Balancing course access and campus efficiency is a challenge and requirement for a class scheduling system when 36% of entry-level courses are packed with enrollment at 95% in public institutions (Smith, 2016).

3.2 Using the Conflict Matrix

The conflict matrix created by MAPPER shows the classes planned for a semester and the conflicts by class for students planning to take the classes. The interpretation of the conflict matrix is done by selecting a class, see Table 1 in the appendices, for example: select CIS 205, the numbers below the asterisk (*) show the number of students in the classes on the lines below that are also MAPPED (planning) to take these other classes. Therefore, of the students planning to take CIS 205, there are 5 also planning to take CS 203, and 6 in CIS 205 are planning to take IS 350. Additionally, the conflict matrix indicates the number of students mapped for a class; for example, CIS 205 has 30 students, shown as (30), followed by a simple code displaying the semesters the class if offered, FWS means Fall, Winter, and Spring, then the name of the class.

3.3 Scheduling Classes Pre-Automation

Pre-Class Scheduler, using the conflict matrix from MAPPER was a manual operation. The seven CS, IS, and IT faculty members would query MAPPER for a current Conflict Matrix and plan a semester with 28 classes on a white board, this process would take about 2 hours. Colleagues, program leads and department chairs at the same university employ various methods to schedule classes, including spreadsheets, whites boards, and floating sticky notes. As the champions of teaching automation to increase efficiency we felt the need to practice what we preach and abandon the white board for an automated solution (Fox, 2012).

3.4 Requirements for the Class Scheduler

After teaching RoR as an upper-level class for CS students, an idea was born to develop an automated scheduling system, a drag-and-drop online interface that would allow a user to select classes, class locations, times, days, and instructors. The system would also allow the user to color code the different instructor objects.

A requirement for the new system to import conflict matrix data from MAPPER was necessary; additionally, the new system should display conflicts as classes are dropped on a time slot. Conflicts would need to be clearly displayed, showing the number of students planning to take both classes. Simply moving the class object to another time slot or offering two sections of the class could remove the conflict.

4. METHODOLOGY

The class scheduler system idea was created while scheduling our classes. Instead of using a whiteboard, and erasing and adding classes to time slots, we thought it would be more efficient to have a digital application with a drag-and-drop interface so we could easily plan a semester of classes. We needed to follow the elements of Agile development to satisfy the needs of the customers (our department) with constant feedback, accept requirement changes on the go at any stage of development, provide constant feedback to our customers, and finally, test the system as each new feature was coded (Hneif, 2009).

4.1 Creating Version 1 in Java

A simple Java application was developed and used during our next scheduling meeting. But, we still had two problems. First, we needed to look up scheduling conflicts in MAPPER manually and make sure our students could take all of their required classes without conflict. Second, our application did not persist the schedule into a document or database that could easily be shared with the members of our department.

4.2 Feature Implementation with RoR

By creating a web application with Ruby on Rails, we provided access to all members of our department and delivered a system that provided productivity with extensive reuse of software (Fox, 2012).

As development continued, it followed a lonely version of Agile, coined Agile Solo (Nyström, 2011), and developed by Watts S. Humphrey in 1993, he used the phrase Personal Software Process (PSP). In this process, a single developer follows an iterative process of planning, development, and postmortem. Development included several steps: requirement, design, coding, and testing. Although a single developer followed the development process in our case, the other members of the department were included as users in the planning and requirements steps.

The next version of the class scheduler was developed to persist the data into a database and to also include the scheduling conflict data from the legacy MAPPER system. The database has semesters, instructors, courses, terms, and periods (days and times), Using Ruby on Rails, we developed the class schedule so that we can create, update and delete all of these entities. An additional entity, called an offering, is able to connect to a semester and course to an instructor, room, and period through database relationships.

For each semester, we create an offering by dragging a course from a list of all courses, and dropping it into a list of offerings (from left to right, Appendices, Figure 4). From there, we assign the course to an instructor, which color codes the course so we can easily see what each instructor is teaching when looking at a semester schedule.

We then move to a scheduling screen that lays out a blank schedule matrix with rooms along the top as columns and time slots as rows. On the left side of the matrix is a list of courses that still need to be scheduled. By dragging and dropping courses into time slots, we schedule classes. Figure 5 in the Appendices displays the Class Scheduler view that allows the adding of periods (class meeting times).

To process the conflict data from the legacy MAPPER system and to store this data, we create another entity called a conflict. Each conflict is connected to a semester and two course offerings. It also contains the number of students that want to take both courses that semester.

As each class gets scheduled, Class Scheduler queries the conflict table to see if any two classes scheduled for the same time period have any conflicts. We display the conflicts on the right side of the scheduling matrix, see Figure 1. In this case, part of an actual semester schedule, all the conflicts are minimal, since a CIS 200 is offered at two different time slots and also offered as an online course as well (not shown).

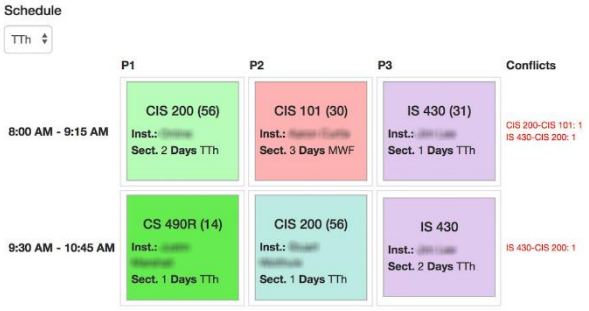

Figure 1, T/TH class conflicts, Fall 2019

"CIS200-CIS101:1" is interpreted as these two classes have one conflict. Figure 2 shows a more complex set of conflicts, with "IT 320-IS 350: 9" indicating that 9 students have a conflict if they schedule these two classes together. These conflicts were avoided by scheduling multiple sections of CIS 350. In both Figure 1 and Figure 2 the conflicts were resolved with multiple sections, but this is not always possible, most of our classes, 13 of 20, have single sections. In Figure 3 in the appendices, there are no conflicts between classes with one section, the Class

Scheduler showed such conflicts during the scheduling process and classes were moved around until there were no conflicts between classes with one section.

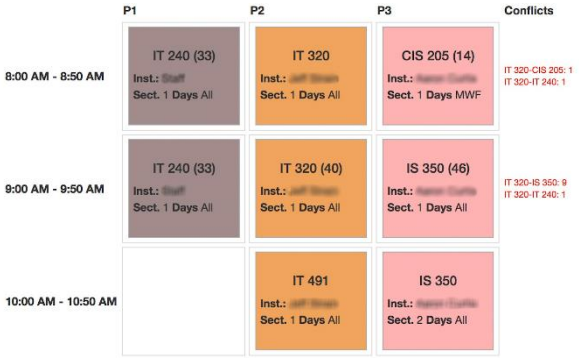

Figure 2, MWF class conflicts, Fall 2019

4.3 Implementing Conflict Data

The class scheduler needs a way to import the conflicts stored in the legacy MAPPER system. To do this, we copy and paste the conflict matrix from MAPPER into a text area within the class scheduler. In Figure 6, we show actual conflict data, the figure is truncated, as the actual list contained 100 conflict records. The class scheduler parses the conflict matrix and collects the number of conflicts between each pair of courses. The class scheduler matches the courses from MAPPER to courses stored in its own database by name. The conflict import is done after offerings are made for each semester and before classes are scheduled. Figure 2 in the appendices shows a completed semester schedule, which has employed the conflict data for that semester. As a note, a key to the success of this system is the fact that our advisors work diligently to make sure student maps are up to date and contain current class schedules.

5. SUMMARY AND CONCLUSIONS

Creating a system that exemplified the principles taught in a RoR class increased class scheduling accuracy. Additionally, the time required to schedule was reduced significantly, most recently, 3 semesters (29 classes, 29 classes, and 17 classes) were scheduled in less then 2 hours.

Planned future research and development activities include:

1. Improve the class scheduler system to automatically import conflicts by "screen scrapping" MAPPER's output.

2. Improve the user interface to allow for more rooms to be scheduled, this will allow other departments on campus to use Class Scheduler.

3. Add the ability for Class Scheduler to automatically schedule classes in ERP systems.

4. Research further the state of legacy systems to quantify and describe the extent of data sheltered in legacy systems.

6. REFERENCES

Ad Astra Information Systems (2016, 2018). The Higher Education Scheduling Index. Retrieved 10 June 2019 from https://thehighereducation.com/

Fox, A., & Patterson, D. (2012). Crossing the Software Education Chasm. Communications of the ACM, 55(5), 44-49.

Hanover Research (2018). Best Practices in Course Scheduling. Retrieved 10 June 2019 from https://www.hanoverresearch.com/

Hneif, M., & Ow, S. (2009). Review of Agile Aethodologies in Software Development. International Journal of Research and Reviews in Applied Sciences, 1(1), 1-8.

Martens, A., Book, M., & Gruhn, V. (2018). A Data Decomposition Method for Stepwise Migration of Complex Legacy Data. In IEEE ICSE-SEIP '18 Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, 33-42.

Meenakshi, S. (2015). Ruby on Rails – An Agile Developer's Framework. International Journal of Computer Applications, 112(1).

Ngoc, V. (2015). Teaching and Learning Scheduler System. Vietnam National University, Hanoi, University of Engineering and Technology, Hanoi.

Nyström, A. (2011). Agile Solo-Defining and Evaluating an Agile Software Development Process for a Single Software Developer. A Nyström - 2011 - publications.lib.chalmers.se

Paulson, L. D. (2007). Developers Shift to Dynamic Programming Languages. Computer, 40(2), 12-15.

Powner, D. (2016). Information Technology, Federal Agencies Need to Address Aging Legacy Systems. Testimony Before the Committee on Oversight and Government Reform, House of Representatives. Retrieved 10 June 2019 from http://www.gao.gov/

Scharlau, B. (2007). Teaching Ruby on Rails. Conference Proceedings of the Java and

Internet in Computing Curriculum Conference in London Metropolitan University. 24 - 29.

Smith, A. (2016). Fixing Capacity With Better Class Scheduling. Retrieved 10 June 2019 from https://insidehighered.com/

Thiran, P., Risch, T., Costilla, C., Henrard, J., Kabisch, T., Petrini, J., ... & Hainaut, J. L. (2005). Report on the Workshop on Wrapper Techniques for Legacy Data Systems. SIGMOD Record, 34(3), 85-86.

Thomas, D. (2008). Enabling application agility: Software as a Service, Cloud Computing and Dynamic Languages. Journal of object technology, 7(4), 29-32.

Tiobe (2019). Tiobe Index for June 2019. Retrieved June 12, 2019 from https://www.tiobe.com/tiobe-index/

ISSN: 2473-4901
v5 n4927

Appendices

```
*=CIS 101 (101) (FWS) Beginning Programming
8 *=CIS 200 (80) (FWS) Fundamentals of Information Systems and Technology
2 6 *=CIS 202 (46) (FWS) Object-Oriented Programming I
1 1 + *=CIS 205 (30) (FWS) Discrete Mathematics I
1 1 . 2 *=CIS 206 (38) (F) Discrete Mathematics II
1 . 2 3 + *=CIS 305 (45) (FW) Systems Engineering I
. . . . 1 . *=CIS 470 (22) (FS) Ethics in Computer and Information Sciences
1 4 . 5 + + 1 *=CS 203 (60) (F) Object-Oriented Programming II
. . . . 1 4 8 1 *=CS 401 (29) (F) Web Applications Development
. . . . 1 4 8 1 + *=CS 415 (26) (F) Operating Systems Design
. . . . . 1 5 . + + *=CS 490R (18) (FWS) Advanced Topics in Computer Science
2 3 4 6 + + . + . 1 2 *=IS 350 (62) (FW) Database Management Systems
. 3 . . 4 3 4 4 2 . . 4 *=IS 430 (36) (FW) Foundations in IT Services, Enterprise Systems, and ERP Skills
. . . . 3 . 4 . . . . . 5 *=IS 485 (18) (FS) Project Management and Practice
5 3 . 1 1 1 1 . 1 1 3 1 . *=IT 224 (37) (FWS) Computer Hardware and Systems Software
6 2 . 1 1 1 . 1 . . . 4 . . + *=IT 224L (25) (FWS) Computer Hardware and Systems Software Lab
+ 8 6 . . 1 1 3 . 1 1 1 . . 6 4 *=IT 240 (57) (FWS) Fundamentals of Web Design and Technology
7 5 7 4 3 1 . 4 1 1 . 3 . . 7 5 4 *=IT 280 (32) (FWS) Computer Networking
3 3 5 4 . + 2 1 2 2 1 + 1 3 5 5 1 5 *=IT 320 (45) (F) Linux Essentials
. . 1 . 3 6 + 1 2 . . 2 8 + . . . . + *=IT 426 (35) (F) Computer Network Services

. means zero, + means 10 or more
```

Table 1: Conflict Matrix for All Students (2195) Fall 2019

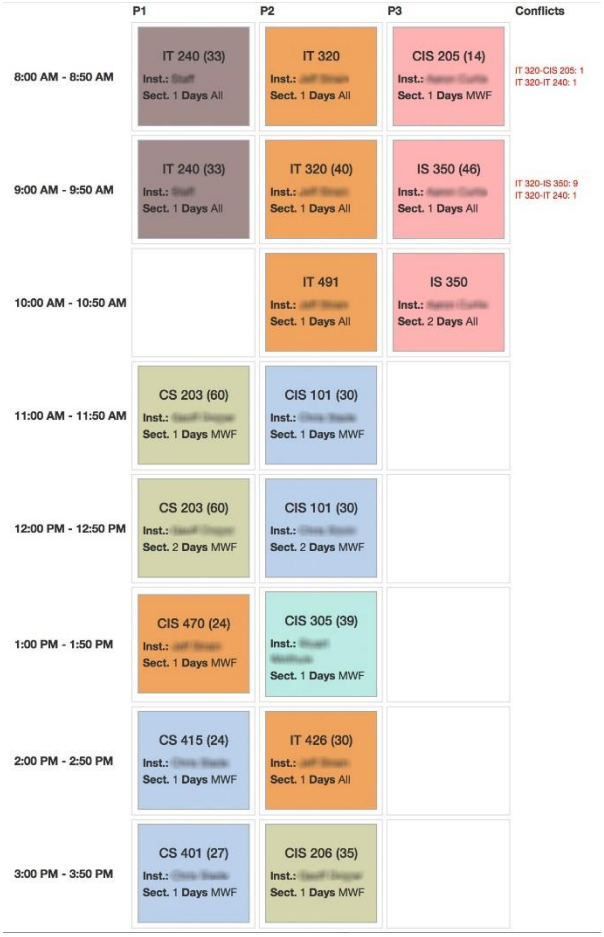

Figure 3: Class Scheduler, Completed Semester from Schedule View

ISSN: 2473-4901
v5 n4927

ISSN: 2473-4901
v5 n4927

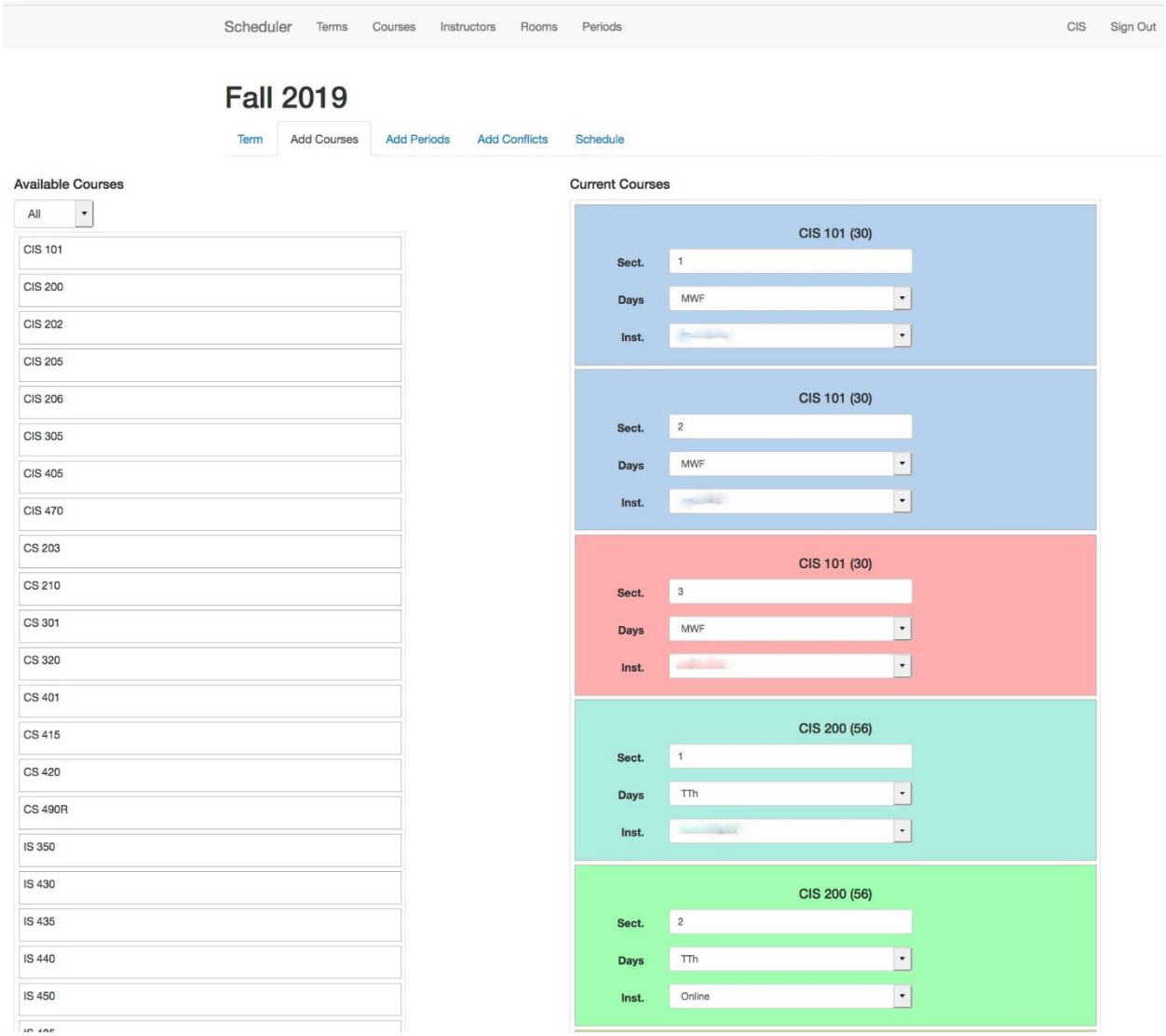

Figure 4: Class Scheduler Add Courses View

ISSN: 2473-4901
v5 n4927

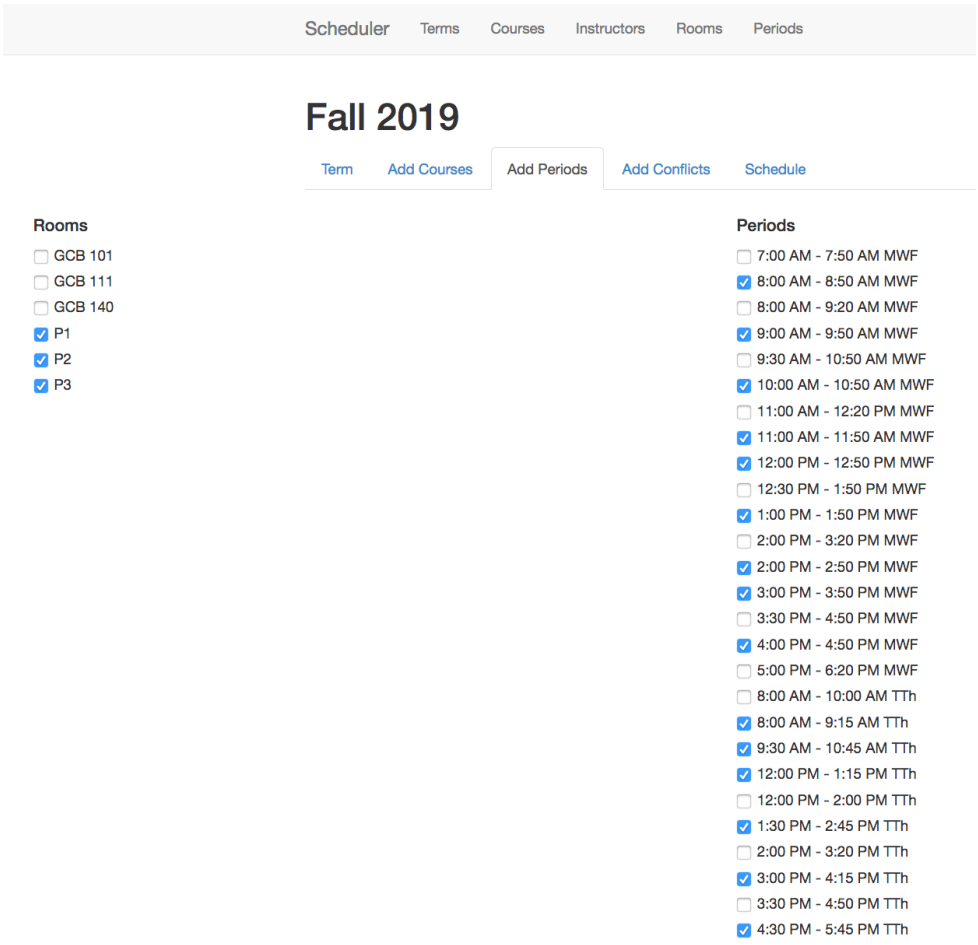

Figure 5: Class Scheduler Add Periods View

ISSN: 2473-4901
v5 n4927

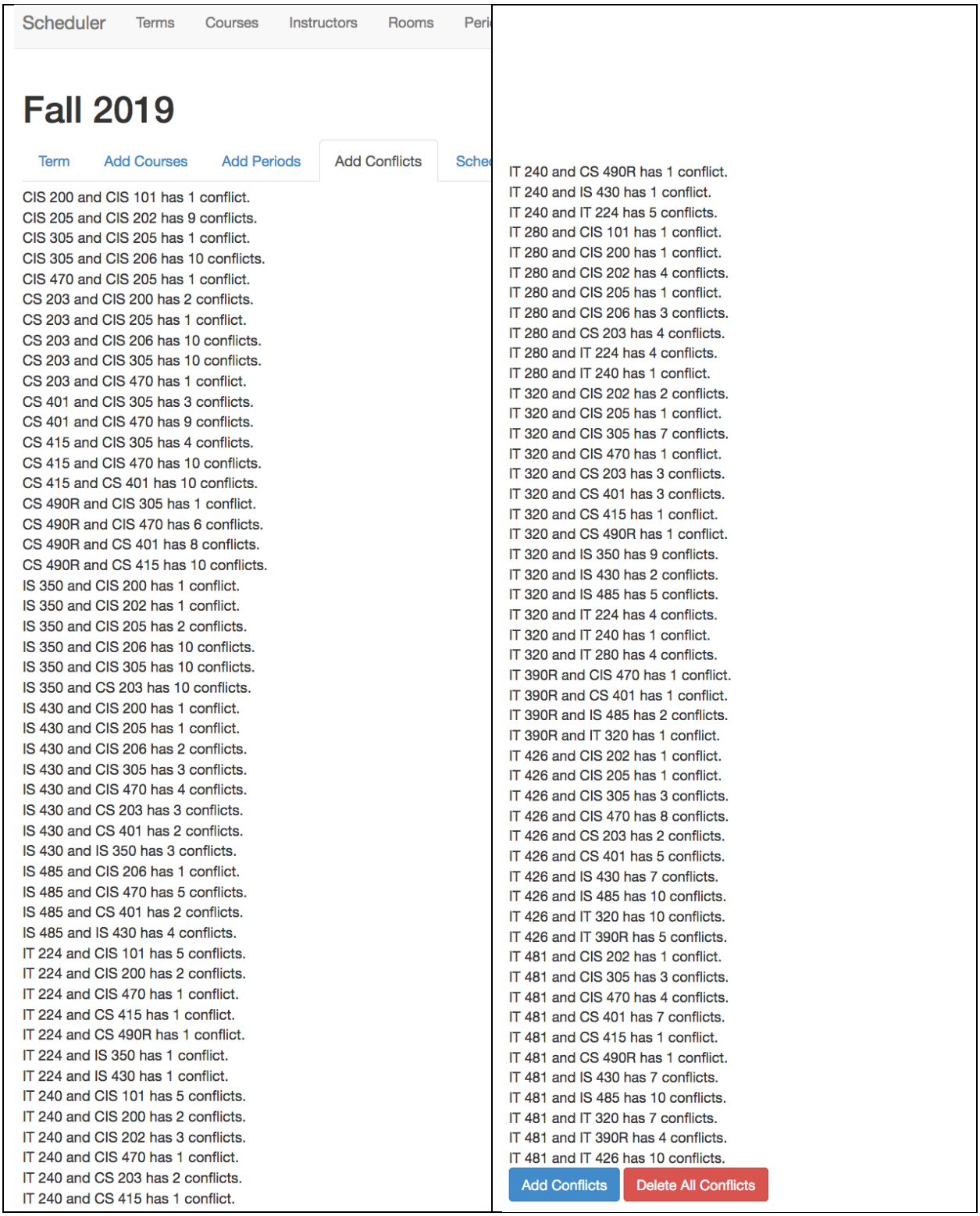

Figure 6: Class Scheduler Add Conflicts View