Applying Contextual Design to Build a Course Scheduler: A Case Study

Jean D'Amico, Teresa Hübscher-Younger, Roland Hübscher and N. Hari Narayanan
Dept. of Computer Science & Software Engineering
Auburn University
Auburn, AL 3849-5347
{damico,teresa,roland,narayan}@eng.auburn.edu

ABSTRACT

This paper describes the design of a course-scheduling tool that assists professors in planning events such as tests, assignments and lectures. It takes university calendar information along with user inputs of course events, and translates these into algebraic constraints, which are then solved by a linear programming algorithm to generate a feasible schedule. We report on the application of the Contextual Design [1] methodology to generate requirements and architecture of this tool. First, a contextual analysis was conducted to ensure that specifications were adequately developed. This included conducting and coding interviews with potential users, and gathering and analyzing work artifacts. Then a prototype was implemented and evaluated by means of a cognitive walkthrough. By describing the design process and the resulting system, this paper makes two contributions. First, it illustrates the contextual approach to interactive system design in an educational domain. Second, it presents the architecture of a course-scheduling tool, with a constraint-solver based on the linear programming algorithm at its heart, which allows the automatic creation of feasible event schedules.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Design Tools and Techniques – *user interfaces.* H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *evaluation/methodology, user-centered design.*

General Terms: Design, Human Factors.

Keywords: contextual design, interactive systems, scheduling, constraint satisfaction, linear programming.

1. INTRODUCTION

Planning and scheduling the various events involved in a semester-long college course can be a tedious and time-intensive task. This is especially true if the professor has several constraints that the schedule must meet. For example, he may need one week to grade assignments, wish to give one week to students for doing each assignment, and plan to have two assignments assigned, graded and returned at least one week prior to the first exam, which has to take place no later than the fifth week of classes. Clearly, the more events and constraints there are, the more tedious manual scheduling becomes.

This paper describes a project on developing a course-scheduling tool to assist professors. Our goal was to develop a tool that professors can use to create and examine different feasible schedules that conform to given constraints, once the course events that the professor wants to schedule have all been specified. This meant that the tool had to be interactive, allowing the input of events and constraints, the computation and display of feasible schedules, and subsequent examination, selection and modification of the schedules. Given this fact, and that there exists major individual differences among professors in how they schedule course events, we chose to employ a design method known as Contextual Design. Contextual Design is a method for developing products that focuses on how the user performs tasks within the context of the work environment itself [1]. This approach was employed to (1) develop a detailed picture of the process by which professors plan a college course and schedule its activities and the factors that influence this process, and (2) subsequently derive system requirements from this picture.

The resulting system, called the Constraint-Based Course Building Tool (CB2T) uses a series of user-defined and internal system constraints in order to assist in course scheduling. These constraints consist of dates (specifically course start and end dates, break start and end dates, holidays, and mid-semester) as well as the number of events (tests, assignments, lectures, and extra-credit assignments) that will occur during the semester. It uses a linear programming algorithm to determine a class schedule that minimizes conflicts between events and allows for adequate spacing between events. The goals of this paper are to describe the design process, to serve as a case study in the application of the Contextual Design technique, and to present the architecture of this system.

2. CONTEXTUAL DESIGN

Contextual Design is a method for developing products that focuses on how the user performs tasks within the context of the work environment itself [1]. In our case, it meant looking at the process of course development as it exists within a university culture by gathering information from people who were potential users of the tool. It was crucial to fully understand the process and meaning of "creating a college course" from the viewpoint of these potential users. It was also important to

Proceedings of the 41st ACM Southeast Conference, pp. 330-336.

understand the essential attributes of a course and the course planning process.

The first step in the information-gathering process was interviewing potential users of the CB2T system. The purpose of these interviews was to determine how a professor developed a course plan, how a syllabus was created and used, what role calendars played in course development, and other time management and course planning issues. The interview process consisted of a short (less than 30 minutes) informal discussion about what types of planning go into creating a college course. We conducted five interviews with faculty members known to have different teaching styles. We also collected and analyzed

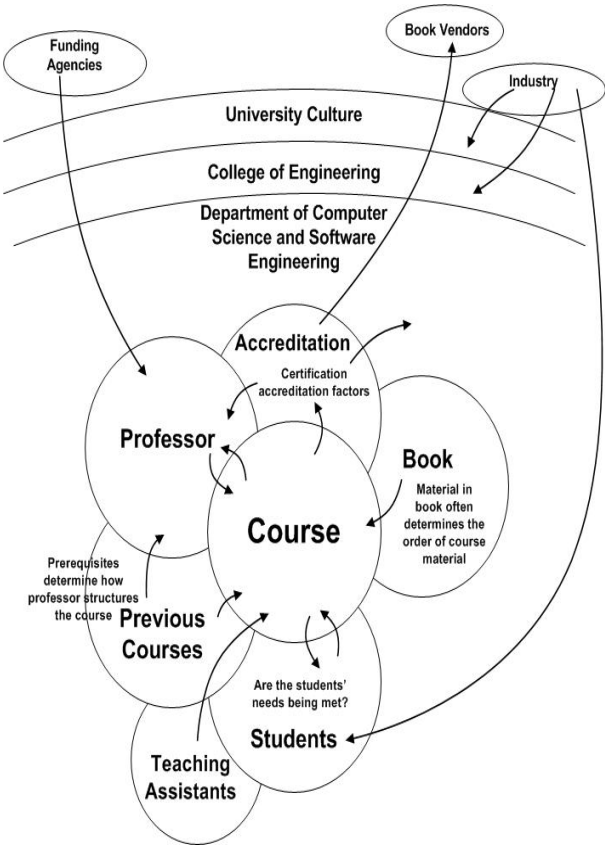

Figure 1. Cultural model.

artifacts used in planning and managing courses that further demonstrated their procedure; primarily calendars, syllabi, websites, schedules, and other related materials. The information gathered was then coded and compiled into useable models. Coding is a process that is used in qualitative research to extract relevant data from observations [3]. Coding schemes allow the data to be categorized and managed into useful information themes.

The Contextual Design process involves building a series of work models that are used to characterize the task that is being completed. The Cultural Model investigates the overall culture of the workplace. The Sequence Model is used to show how a sequence of events unfolds over time. The Artifact Model looks at how actual objects are used in the work process. These three models were developed.

The Cultural Model (Figure 1) encoded how various people, institutions, and influences impacted the course planning process. Obviously, professors and students most heavily impact a college course. Beyond these influences there are other, less obvious, components that impact a course. For instance, there is the immediate influence of the department and the university on the class. They influence how the class is structured through factors such as the schedule for the school year, vacations, and the criteria by which the course will be evaluated. Other factors include accreditation boards, who determine what information should be covered in a course; the book vendors, who often impact which book is chosen which in turn determines the sequence of the material to be covered; teaching assistants, who influence how quickly grades are returned to students and the course load the professor is able to assign; and industry trends, which impact, over the long-term, the types of courses that are taught in universities.

The Sequence Model is used to show how a sequence of events unfolds over time. From the interviews, we could determine a generalized method that professors were using to develop courses. This series of events was broken down into three main parts: (1) Developing initial course material; (2) Developing a syllabus and class schedule; (3) Teaching the class and managing/modifying course content. Developing initial course material involved such things as researching and choosing the appropriate book, readings and other materials relevant for teaching the class. From this information users would typically develop a set of requirements (either formally or informally) in order to determine and schedule a combination of activities that are most appropriate for teaching the material, such as lectures, quizzes, in-class exercises, tests, and assignments.

Different professors used various approaches to determine the dates for scheduling various activities. The Sequence Model outlined three different approaches that could be taken regarding event scheduling: (1) Events such as tests occur after a given period of time and are fixed; (2) Events occur after a given period of time but can be moved within reason; (3) Events occur after a given amount of material has been covered. The final step of the Sequence Model outlined how professors would perform routine maintenance on a course. This included making daily changes to the schedule and format of the class for teaching the same course at a future time.

The Artifact Model was created by analyzing artifacts collected from the faculty members who were interviewed, coding this information, and creating a single document that explicated the purpose, characteristics and components of the three types of artifacts that we discovered were being used by the interviewed faculty: planning documents, syllabi and calendars. These documents brought to light a great deal of information that was often not available directly in the interviews, besides illustrating how professors tangibly laid out their courses. For instance, one planning document was a flow chart of a step-by-step iterative process that thoroughly described how a course schedule was constructed by a particular professor. The resulting model thus

provided additional insights into how a typical user carried out the task under analysis.

In particular, planning documents revealed interesting individual differences. Some professors, when building a course, use a highly structured method in which every class period is planned out at the beginning of the semester and leaves very little margin for change within the schedule. Such professors develop what we call a static calendar. It is developed at the beginning of the semester and defines all lectures, assignments, tests, and extra-credit assignments. It is often included as a part of the syllabus. Other professors take a more flexible approach in which the topics are described generally within the syllabus, which provides a general outline for the course. They create what we call a dynamic calendar, which is actually a schedule of events that is created with the deliberate intent of being altered as necessary during the semester. Such a calendar might schedule specific events at the beginning of the semester, but with the implicit understanding (by professors and students) that the events may be moved as the course progresses.

3. REQUIREMENTS SPECIFICATION

The steps of user interviews, artifact collection, information analysis and coding, and model development yielded several system requirements.

Requirement 1:The system should include daily, weekly, monthly and semester calendar views of the course plan, with a provision for entering and flexibly displaying (i.e. with more or less details) event information such as title, time, location and description. One interesting aspect revealed by the artifacts was that types of calendar views the professors created took many different forms. Some professors would use a weekly or monthly calendar view with only the event name and title (i.e. lecture 12 or exam 1). Other professors would use a daily view that would show the details of every class event. Still other professors would just have a course outline that showed the sequence of topics to be covered with no details, defined events, or dates.

Requirement 2: The system should support the creation and maintenance of private (can be seen only by the user) and public (can be made publicly viewable, say, over the web) calendars, with facilities for easy migration of events from the private to the public calendar. The Artifact Model indicated that a calendar or schedule could either be a privately held document by the professor alone or a public document that is shared between the students and professor. In many cases, professors will keep two separate documents: a public one that is openly available to the class and a private one that is used for planning and revising before adding items to the public one. The private calendar may contain course information that is intended for the personal use of the professor and not for consumption by the class at large. Some events first appear in the private calendar and later appear in the class calendar. Further analysis revealed that this is an indication of background planning and revision that takes place before information is posted publicly.

Requirement 3: The system should allow the user to hide and show course activities by type and by detail. The interviews and the Artifact Model indicated that users would like to view their daily, weekly or monthly schedules in different ways based on type and desired level of detail. For instance, sometimes they may want to see all tests that are scheduled for the current day, week or month. They may also just want to see the titles of all lectures scheduled for a week or see the detailed descriptions of all lectures scheduled for the day.

Requirement 4: A course calendar should be easily changeable at any time. The Sequence and Artifact Models indicated that the content and timing of course events are often changed as the semester progresses, sometimes affecting the current offering of the course and sometimes with an eye toward the next offering of the course. These maintenance activities indicated the need for highly customizable course content while the course is being taught.

Requirement 5: Allow course plans to be saved as reusable templates. From the Cultural Model we identified the impact of previous offerings of a course on its current content and schedule. This indicated the desirability of a template feature that would allow course information to be saved as a template for future modification and/or reuse.

Requirement 6: The system should allow the user to create and maintain multiple calendars for the same semester, and calendars of the same course over several semesters. The Sequence Model indicated that professors created and maintained separate plans for all the courses they were teaching, and also revised and maintained course plans over several semesters.

Requirement 7: The system should separately maintain data for each user and each course. Since it is common for the same course to be taught by different professors at the same or different semesters, and for the same professor to teach separate courses in any given semester, the data need to be kept separate and access needs to be controlled through a password-protected user login facility.

Requirement 8: The system should maintain an easily searchable and printable database of all course information. The Cultural Model also highlighted the importance of accreditation concerns to course planning. Typically, an accreditation board will visit a university and ask to see syllabi and other course related material from the courses that have been taught since the last certification. So a system that captures and archives the complete schedule, syllabus and content (i.e. topics covered in each lecture) from multiple offerings of a course in a uniform format can be invaluable in preparing reports for accreditation boards.

Requirement 9: The system should not enforce a single approach to planning and scheduling; instead, it should be as flexible as possible. The Sequence Model revealed an inherent difficulty in developing a tool like CB2T: there are large individual differences and no single approach fits everyone. The Artifact Model suggested that a course schedule must be flexible and dynamic, to be used in different ways depending on the nature of the user. For example, detailed planners construct a complete and detailed schedule at the beginning of a semester and adhere

to it as much as possible through the semester, but flexible planners construct just an outline at the beginning and fill in details and dates as the semester progresses. In order for a system like CB2T to fit the variety of approaches that people could potentially use, this flexibility requirement needs to be further refined as follows. (9.1) The system should allow a user to input as many, or as few, course events and constrains as he/she wishes. (9.2) It should allow the user to choose between automatic scheduling of the events entered or manually constructing a schedule. (9.3) For automatic scheduling, instead of generating and showing the "optimal" schedule or a single feasible schedule, the system should support the user in developing a schedule that best fits his or her approach to course planning. (9.4) This means that the system should support an iterative approach to scheduling: (9.4.1) Calculate and display all or several feasible schedules; (9.4.2) Then allow the user to choose (and modify if desired) one of these schedules; (9.4.3) It should also allow the user to reject all offered schedules and instead change some of the events and constraints for the next iteration.

Requirement 10: The system should incorporate, and allow the user to enter and modify, a variety of constraints or relationships among various course events. The Artifact Model provided information regarding several types of constraints that are applied during course planning. First, there are hard constraints. For example, no formal course event can be scheduled on a Sunday or a university holiday. Similarly, universities generally require final exams to be held on specific dates and times. Second, there are soft constraints. For example, a professor may want the initial schedule to meet a constraint that the default length of time to grade an assignment is a week. Another example is that an assignment due date cannot be scheduled within a week of a test. These are soft constraints because they are user-defined and therefore can be modified or relaxed by the user. Given this distinction between hard and soft constraints, this requirement can be further refined as follows. (10.1) The system should automatically acquire hard constraints such as the start and end dates of the semester, scheduled holidays, final exam schedule etc. (10.2) The system should allow the user to enter additional soft constraints and modify or relax them as needed during iterative scheduling.

Requirement 11: The system should support manual and automatic emailing of students with course information and reminders about upcoming deadlines. The interviews revealed that this is a highly desired functionality that paper-based calendars cannot provide.

4. SYSTEM ARCHITECTURE

Based on the requirements generated from the Contextual Design process, we developed a three-level architecture for CB2T (Figure 2). The highest level is a PHP/HTML user interface level that presents information to, and collects information from, the user. At the next level, a Java program translates this information into a linear program which is then solved by a constraint solver at the lowest level. Once the solver is done, the middle level interprets the output of the constraint solver and translates it into a form that can be used by the interface level. All of the data that is used throughout the levels are stored and retrieved by a MySQL database.

A prototype of this architecture has been implemented. Feasible schedules are successfully generated in real time.

4.1 User Interface

The user interface provides five main functionalities: user login, creating a course, entering constraints, viewing a course schedule by day, week or month, and creating, editing and deleting individual course events. In order to create a new course, the user can enter the following information: course title, instructor name, instructor email address, instructor telephone number, instructor office location, instructor office hours, GTA name, GTA email address, GTA office location, GTA office hours, course description, location (classroom and building), and lecture time. They can also input (implicit) constraints: semester begin date, semester end date, holidays, and any breaks (e.g., spring break), class rotation (Monday/Wednesday/Friday or Tuesday/Thursday), the number of assignments, the number of extra-credit assignments, and the number of tests. From this, the system computes information

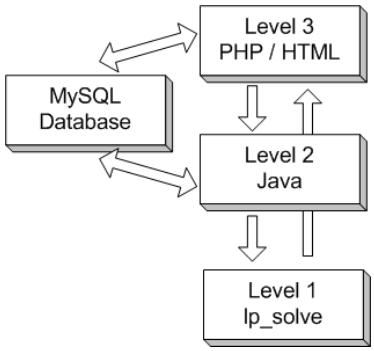

Figure 2. System architecture.

such as the actual number of lecture days in the semester.

The system defines four types of events. An assignment is an event that is defined as a class activity with a due date attached to it. A test is an event that occurs during a lecture on a date assigned by the professor. An extra-credit assignment is an optional assignment. Lectures are hour-long events that take place during the class period. These occur every class day with the exception of days that tests are scheduled. Events have associated features such as type, name, date, description and notification schedule (i.e. to schedule automatic email reminders for students). The user can manually add a new event and edit or delete an existing one at any time. The system automatically checks each new or modified event against the course calendar for a variety of conflicts. Conflicts include events being held on weekends, holidays, breaks, or overlaps with existing events. If a conflict is detected, the user is asked to modify the conflicting feature. After all conflicts are resolved, the event is added to the calendar.

The course calendar is viewable by day, week, or month. All views, by default, show events associated with all courses for the current semester created by the current user. The user can toggle individual courses on and off, thereby restricting the

views to events associated with a specific course or courses. The default view is the current day view. Events can be clicked on to bring up full descriptions.

4.2 The Translator

The Java program at the middle level takes the specification inputted into the PHP/HTML form as input from the interface level using shell scripting through PHP and translates them from domain specific constraints as described in the previous section linear inequalities as follows. First, more general domain constraints specifying the length of the semester, holidays, etc. are added to the user's specification. Then, all constraints are translated into a set temporal constraints which, in turn, are directly translated into a linear program. This two-step translation allows reduces the translation to two simple steps. An example will better clarify this translation process.

Suppose a user enters the following course information: Course dates: August 19 2002 - December 6 2002; Holidays: September 2 2002; Break: November 25 2002 - December 1 2002; Schedule: MWF; Tests: 2; Assignments: 2; Extra-credit Assignments: 0; Mid-term date: October 9 2002.

The total number of days between the start and end date, 109, is first calculated. Then a determination of which are weekdays and which are actual class days (Mondays, Wednesdays and Fridays excluding holidays and break) is made. Then this level constructs algebraic equations representing constraints implicit in the information entered by the user and several predefined system constraints. One predefined system constraint is that a test and an assignment event cannot occur on the same day. This results in 4 inequalities of the form $Ti\text{-}Aj \bullet 0$ where Ti is the day of the i^{th} test and Aj is the day of the j^{th} assignment. The system also has predefined constraints of the form event-i must occur before event-$i+1$ for any event type. This generates inequalities $T1\text{-}T2 < 0$ and $A1\text{-}A2 < 0$. Another predefined constraint is that the first test cannot occur within the first seven days of the semester. This can be expressed with $T1 > 7$. A fourth constraint is that assignments need 7 days for students to do them and for the professor to grade. A fifth constraint is that at least one assignment must precede a test. A sixth predefined system constraint is that any outstanding assignments must be graded and returned at least 7 days before a test. These three constraints give rise to the inequality $A1+21\text{-}T1 < 0$. A seventh constraint is that a test must be given and returned after grading to students at least a week before the mid-term, which is the last possible date for dropping a course. In the present example, the mid-term is day 51 out of a 109-day semester. This yields $T1+14 \bullet 51$ requiring that test 1 must occur at least 14 days prior to the mid-term.

Variables of the form *eiej* stand for the number of days between event-*i* and event-*j*, where the letter *e* is replaced with *S, E, T, A, L* or *X* to represent the start of the semester, the end of the semester, a test, an assignment, a lecture or an extra-credit assignment respectively. This, in the present example, results in six constraints $T1\text{-}S\text{-}ST1 \bullet 0$, $T2\text{-}T1\text{-}T1T2 \bullet 0$, $E\text{-}T2\text{-}T2E \bullet 0$, $A1\text{-}S\text{-}SA1 \bullet 0$, $A2\text{-}A1\text{-}A1A2 \bullet 0$ and $E\text{-}A2\text{-}A2E \bullet 0$. Note that in our example $S = 1$ and $E = 109$. An eighth predefined constraint is that tests be equality distributed over the semester. This gives rise to the equations $ST1 = T1T2$ and $T1T2 = T2E$. The constraint solver will attempt to maximize these values subject to their sum being less than or equal to the total number of days in the semester. The translation level passes this set of equations to the constraint solver at the lower level. It will solve these constraints and return the corresponding variable values back to the translation level.

A solution for the present example is $A1 = 7$; $A2 = 58$; $T1 = 37$; $T2 = 73$; $SA1 = 51$; $A1A2 = 51$; $A2E = 51$; $ST1 = 36$; $T1T2 = 36$; $T2E = 36$. These values stand for the number of days from the start of the semester (i.e. test 1 is scheduled for the 37^{th} day). The translation level translates these numbers into actual dates and checks them to ensure that they do not fall on Saturdays, Sundays, non-class days, holidays, or breaks (if they do, the closest class days are picked instead). Then these are written into the database and passed to the interface level for display. Subsequently, the user is presented a screen that shows the events that have been scheduled with the option to edit each by clicking on it.

4.3 The Constraint Solver

Generating feasible schedules by developing and solving constraints is a feature of CB2T that separates it from commercially available course management tools such as WebCT. The Artifact Model indicated that professors created course schedules from relatively few constraints, notably the number of tests, assignments, and extra-credit assignments. Constraints such as course start and end dates, holidays, break start and end dates are the same for all courses and are automatically added to the set of constraints. As the example above showed, both implicit (i.e. the semester calendar) and explicit (i.e. relationships among various events to be scheduled) constraints can be expressed as algebraic inequalities and don't have to be treated differently at the computational level. Therefore, we adopted the linear programming approach to constraint solving [4]. We chose an open-source implementation of the linear programming algorithm *lp_solve*.

5. INTERFACE EVALUATION

From the information gathered during the Contextual Design process we constructed profiles of typical users of CB2T. From these profiles we developed use cases and used them in a "cognitive walkthrough" evaluation of the interface. Information gathered from interviews suggested that a "typical" user of the system can be characterized as falling between two extreme kinds of users, whom we call the Schedule-driven User (SU) and the Content-driven User (CU).

SU is a highly structured individual who creates detailed plans. He completely plans his course schedule at the beginning of the semester and follows it closely throughout. He uses a highly formalized process with well-defined steps when creating a course schedule. His first step is to analyze the requirements for the course. After defining the requirements, he refines these into a set of general concepts that all students taking the class should know by the end of the semester. These concepts are translated into specific course objectives. From these objectives, he

determines what topics to teach and how many lectures to give on each. This process is time consuming and tedious, but the end result is a detailed daily lecture schedule for the course. SU would use CB2T to assist him in scheduling events, once course requirements and objectives have been defined and the lectures planned. He is likely to manually schedule course events such as tests and assignments, or at least significantly modify the automatic schedule that CB2T generates for him. However, once the schedule is finalized at the beginning of the semester, he will strictly follow it with minimal modifications during the semester.

CU, on the other hand, is the type of user who describes her course preparation methods as more content-based. She is generally not concerned with planning out her schedule to the minute detail at the beginning of the semester. She is interested in thoroughly covering the course material, and scheduling tests and assignments based on how much gets covered as the semester progresses. Upon deciding to teach a course, she decides what major topics should be covered. She makes only approximate plans at the beginning of the semester, and fills in details later as needed. She wants to have a feasible schedule generated for her by CB2T once she decides on the important course events. This provides her with a view of the semester with tests and assignments fairly evenly distributed throughout the course. It will also provide some insight about the pacing of lectures to cover sufficient material before tests and assignments. As the semester progresses, she will adjust the schedule based on the actual course content she covers from lecture to lecture. Thus, CU is more likely to keep both a private and a public calendar, moving items from the private one to the public one as she finalizes events. She is also the type of user who is likely to require that multiple feasible schedules be generated.

After constructing user profiles, the next step was to develop well-defined use cases to demonstrate how the users would interact with the system and how the system would respond. Use cases are an important part of the Unified Modeling Language (UML). UML, sometimes referred to as the Unified Process, is an object-oriented software design methodology [6]. Use cases provide detailed specifications of actions that the user performs and the system's responses. We developed use cases for the four tasks that all users will perform with CB2T: logging in, creating a course schedule, manually creating a course event and modifying an already created event (see [2] for use case descriptions).

We developed two sets of task descriptions from the use cases, one matching the SU profile and the other matching the CU profile. Both involved logging in, entering information regarding a new course, having a schedule automatically generated, and then rejecting and/or modifying various course events to create a final course schedule. Then we carried out a cognitive walkthrough evaluation by a usability expert. The cognitive walkthrough is an evaluation process that involves constructing task descriptions from a system specification, and then obtaining critical feedback from one or more usability experts who interact with the system to carry out these tasks [5]. While completing the tasks, the expert reviewer attempts to predict how a potential user will view the system and potential problems they may face, and suggests design changes. This allows the interface's usability to be evaluated and improved. Our expert generated several recommendations for improving the interface. Implementing these changes to the interface is part of our ongoing work.

6. CONCLUSION

We presented a case study of the design and evaluation of a course-scheduling tool. It takes university calendar information along with user inputs of various events, and translates these into algebraic constraints, which are then solved by a linear programming algorithm to generate a feasible schedule. We described the application of the Contextual Design methodology to generate models and user profiles, requirements, and the system architecture. The analytical steps preceding the development of models and requirements included conducting and coding interviews with potential users, and gathering and analyzing work artifacts. Then a prototype was built and evaluated through developing use cases and conducting a cognitive walkthrough. Figure 3 summarizes this design process.

By describing this design process and the resulting system, this paper makes two contributions. First, it illustrates the contextual

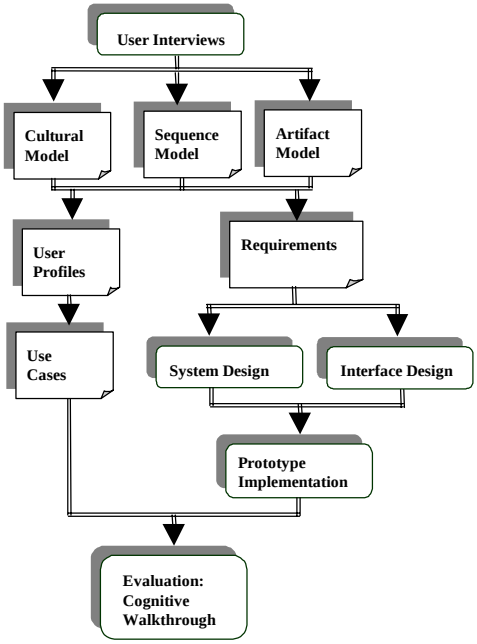

Figure 3. Design process

approach to interactive system design in an educational domain. In particular, it shows how one can explore and formalize the tasks of interest through qualitative data collection and model building. Second, it presents the architecture of a course-scheduling tool, with a constraint-solver based on the linear programming algorithm, which allows the automatic creation of feasible event schedules. Unlike commercial course management tools such as WebCT, CB2T provides a novel functionality – the automatic creation of a feasible event schedule for a course – based on constraint satisfaction.

As what we have described is the first iteration of design, clearly, more work is ahead. The prototype does not yet meet some of the requirements (2, 5, 6, 8 and 10) outlined in Section 3. In particular, a way of ranking and displaying different schedules if there are multiple solutions to the constraints needs to be developed. More sophisticated error checking is also desirable. Future research will focus on these issues. Afterwards, we will be testing actual use by releasing the system to faculty for semester-long usage. Ultimately, we plan to release the system as open-source software.

7. ACKNOWLEDGEMENTS

The support of NSF for this research through grant REC-9815016 is gratefully acknowledged.

8. REFERENCES

[1] Beyer, H., and Holtzblatt, K. Contextual Design: Defining Customer-Centered Systems. Morgan Kaufmann, 1998.

[2] D'Amico, S. J. Constraint-based Course Building Tool. MSwE Report, Computer Science & Software Engineering Dept., Auburn University, Auburn, AL 36849, 2002.

[3] Denzin, N., and Lincoln, Y. (Eds.). Handbook of Qualitative Research. Sage Publishers, 1994.

[4] Fourer, R. (2000). Linear programming frequently asked questions. http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html.

[5] Preece, J., Rogers Y., Sharp, H., Benyon, D., Holland, S., and Carey, T. Human-Computer Interaction. Addison-Wesley, 1994.

[6] Scott, K. The Unified Process Explained. Addison-Wesley, 2002.