The Operations Overtime Scheduling System

A. Chris Eizember

E. I. duPont de Nemours & Co., Inc.
Electronics Development Center
14 T. W. Alexander Drive
Research Triangle Park, NC 27709

Abstract

This case study examines the design and development of an expert system used to fairly distribute the opportunity to earn overtime pay among workers in a manufacturing environment. It describes the problem and the mechanisms used to prioritize workers for selection and details some of the difficulties encountered in the knowledge engineering stage. The benefits derived by using the system are given, as are some of the lessons learned during the development and deployment.

Introduction

The Artificial Intelligence Task Force is a small group within DuPont tasked with catalyzing the use of Expert Systems (ES) throughout the company. To that end, we train 'experts' in the use of ES shells, support them in their development efforts, help them with prototypes, and communicate AI activity throughout the company. We become involved with projects as principal programmers when businesses having an ES need lack the resources necessary for implementation, or when the resulting application may have broad applicability across the company. The Operations Overtime Scheduling System (OOSS) is one such application.

The system assists with the scheduling of manpower with an emphasis on consistently and fairly distributing opportunities to work overtime among the workers. It is in use at a manufacturing facility scheduling some 350 workers and is currently being studied for implementation at other sites.

Problem Description

Scheduling workers in an around-the-clock chemical production environment is a job that never ends. Even with prearranged rotating schedules the task is complicated by meetings which everyone must attend, worker training, unit shut-downs, vacations, and unexpected absences. These schedule upsets and others contribute to the complex job of keeping a full compliment of workers in the area, and making sure that each one is qualified to perform the job that they are there to do.

Oftentimes this becomes a full-time job, with one person juggling lists of weekly schedules, planned overtime, and worker records. They are called upon by supervision to fill the schedule on time, manage worker disputes, and produce summary reports of worked overtime for historical purposes. They are also constantly hounded by workers who want to know 'Why so-and-so got this shift instead of me' and 'Why are my OT hours so high'. Emergencies arise when workers don't show up - they need someone on plant NOW. It can be a very stressful job.

The OOSS was written to help manage all the information required to perform the scheduling. It tracks the weekly schedule, showing an up-to-date representation of who should be working each shift, and what job they will be performing; it also tracks each worker's standing in the overtime hierarchy. These two components are critical in the task of properly filling overtime needs. It accumulates data on the overtime worked in a number of ways; by reason (e.g. to cover a vacation, training, or absence), and by skill; summarized weekly and YTD. This information has been used by management to get 'the big picture' and has allowed the plant to reduce the amount of overtime paid significantly. One other form of information has also been important - the system records all the transactions involved in changing a worker's overtime hours. This has resulted in a direct savings of reducing grievances and an indirect savings of fostering trust in the system.

The OOSS is currently in use at DuPont's LaPorte, Texas manufacturing facility. There are a handful of separate operating units producing chemicals for

different applications but the all share the same rules and procedures for filling schedules and overtime. Any given area will have between 12 and 60 workers on their schedule; workers are not shared between areas. Many areas have one person doing the scheduling tasks (their 'scheduler'); in some of the smaller areas the scheduling is handled by supervisors, a few supervisors even handle multiple areas.

Each area is a separate entity as far as scheduling goes. They have a pool of workers on a 12-hour rotating shift. When workers are 'off' they are eligible to come back to work to fill in for absences or to do extra work. When someone works outside their normal shift rotation they are paid a premium for that time, up to 2.5 times normal pay for working on a holiday. As one might imagine most workers are eager to work at least some overtime as it can constitute a sizable portion of their pay. The problem them becomes how to distribute the overtime fairly among the workers. The priority for selection for filling this extra work is decided by the concept of 'charged overtime hours'.

Each time someone works overtime, their charged OT is incremented to reflect the 'extra' money that they will be paid. When it comes time to select a worker for a shift, they are placed in a ranked list based upon their year-to-date charged OT, and those with the lowest charged OT are offered the chance to work the shift first. This results in a fair method of spreading the extra work among those willing to accept it.

Adding one more level of complexity is the fact that all of the workers in an area are not necessarily skilled in performing all of the possible jobs in that area. Therefore separate information must be kept on exactly which jobs each worker can perform.

The task then becomes one of information management. One must maintain an up-to-date schedule and a list of worker's skills and charged OT. When a shift needs to be filled the system, using the list of workers, eliminates all those who can't work the needed skill and can't work because of schedule conflicts and then produces a 'call-out' list of the remaining workers ranked in ascending charged OT hours. The scheduler then goes through the list calling the workers at home until someone accepts the shift. Then a calculation is made to determine how much time to 'charge' that worker for the shift and the amount is added to their YTD charges.

The real expertise in the system then comes in two areas: selecting the proper workers to offer the overtime to (and in the correct order) and how to do the calculation of the number of overtime hours charged to a worker once they accept a shift. The former was relatively easy to implement, the latter proved to be an exercise in knowledge engineering.

Evolution of an Application

A few of the areas had independently begun to implement their own computer systems to help with the problem, ranging from a spreadsheet used to track hours to a rule-based system used to select workers to call. These systems, and the ways in which they were being used, were beginning to fail under a closer scrutiny of overtime by plant management. Costs for overtime were excessive, rules were being inconsistently applied, and workers were increasingly turning to the union with grievances based upon lost opportunities for extra pay.

A request was made by the plant management to develop a system that would be useful to all the areas (accommodating the varying ways that they implemented policy) and also bring together data to create a picture of overtime costs for the entire plant. The system was to codify the rules under which all areas were operating, and provide enough flexibility so that each area's particular skill set and procedures toward allocating OT could be accommodated.

We began by interviewing people with the expertise that would form the core of the system: how to select qualified workers and how to charge them for the shifts that they worked. What we discovered was that although they were working from the same rules and procedures they approached the problem differently and often came up with different answers. This was no surprise. What we found as we progressed through the development was that as disagreements arose between the experts they would have to discuss the problems among themselves, often returning to the common rules and procedures only to discover that they might have been improperly interpreting the rules.

Application Development

As is the story in the development of many expert systems, the path to a properly functioning system was a learning experience for all involved. Over the course of one and one-half years, three knowledge engineers/ programmers and over ten experts/users were involved in specifying the system, designing the user interface, and elucidating, codifying, and coding the expertise.

Initial meetings revolved around expectations of the various parties, our preferred mode of development (iterative prototyping), system requirements, access to experts, and timeframes.

Tool selection was constrained by the size of the problem, software available, preferred mode of accessing the finished system, and the configuration of the computer system in place. We decided to use RS/1 (from BBN Software Products, Cambridge MA), already on the plant's VAX cluster.

RS/1 provided a framework to store the data (tables), an expert system development environment (RS/Decision), and a general-purpose programming language (RPL). With its structure of shareable group data, information could be made available across workgroups with partitioning between the the various production areas. This also meant that only one copy of the code had to be maintained, and each area could customize the system to meet their needs. (Figure 1)

Using the idea of rapid, iterative prototyping we would work with the experts for a day or so at a time, then return to development to apply what we had learned to correcting and enhancing the system.

Initial development continued for one year before the system was complete enough to be implemented for use in two of the plant's areas. An initial period of parallel record-keeping was planned to continue for two months but after only two weeks the users had enough faith in the system to transition to it completely.

As more areas came on-line, more exceptions in work procedures were found; changes would be made to accommodate them and system evolved more and more flexibility and utility. Approximately 1.5 man-years of effort went into the coding, the experts' time is not included in that figure.

So Where's the AI?

In our experience few simple systems - those consisting of only a small rulebase or decision tree - survive on their own. Successful systems consist of a large part of normal 'computer programming' procedural code assisted in critical areas by AI where appropriate. Additionally, AI, in the form of an expert system tool in this case, is useful in the prototyping stage to rapidly and simply encode knowledge.

The two areas of expertise in this application, selecting workers for OT shifts and charging for the hours worked, were obvious candidates for expert systems. They would be hidden within the larger system - their use would be transparent in the context of using the entire application. Both lent themselves to the rule-based paradigm however neither survived in rulebase form in the final application. Surprisingly, the only 'true' expert-system shell portion of the final system was the on-line help, implemented by a decision-tree.

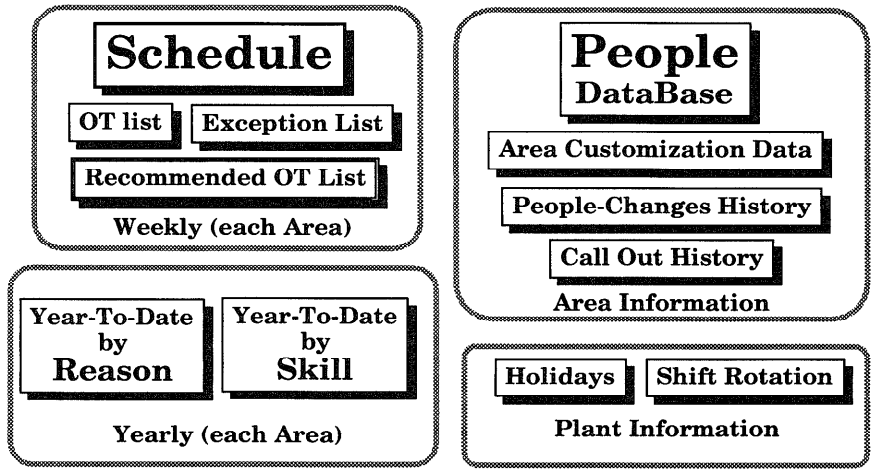

Figure 1: Data Structure

Even though AI didn't survive into the final system in the forms expected, knowledge engineering methods and expert systems shells did simplify the capture of expertise and implementation of the system.

Probably the most complex and confounding portion of the system dealt with the calculations involved in charging for overtime shifts worked. This is where the most disagreements arose among the experts and required the majority of knowledge engineering effort. We started by casting the knowledge into a rulebase. The rulebase's IF - THEN form seemed to match the way the experts approached the calculation. However, the rules started to multiply quickly. There were simple rules for shifts worked on a day outside the normal rotation and more complex rules for working holidays or all seven days in the week. The worst situations happened when exceptions piled up - workers on holidays in the middle of the week when they were working all seven days that week and Sunday had both normal time and overtime. The permutations seemed endless. Complicating matters was the fact that you had to consider the entire week in the calculation, not just the day worked.

Another factor complicating matters was the idea that if a worker initially accepted a shift of overtime then later backed out of it or otherwise didn't work it, the impact had to be subtracted or 'backed out' of their 'charged OT'.

Depending on the situation, the amount of time subtracted wasn't necessarily equal the the amount initially charged. This resulted in separate calculations and rules for both adding and subtracting time.

As we polished the rulebase, adding rules here and there to cover the obscure cases, we realized that although this was the way the experts were explaining things to us, they were, in fact, performing the calculations much differently when asked to do example problems. This resulted in a shift in our interviewing in order to determine how they were doing it, not how they were explaining it.

We had been treating each shift as an individual event - "How many hours would you charge if this person worked in such-and-such situation". Our examples were all for discreet cases. We discovered by considering the impact of a shift on the week - as a whole - we could greatly simplify the situation. By casting the data (the person's weekly schedule complete with overtime shifts) into a table, the rulebase needed to process it shrank to a handful of simple rules, the exceptions that had been posing problems were easily handled. Both adding and subtracting overtime charges could be handled by the same rules. Ultimately the rulebase was coded into a relatively simple procedure with nested IFs. (Figure 2)

```
DO I = 1 TO 7;
    IF ((HOURS_WORKED = 0)and(uns_worked = 0)) OR LOSE_PREM THEN
    {
      TT[I,C_OT] = 0;
     DONEXT;
    }
   CHARGED_OT    = 0;
     SCHEDULED_DAY = TT[I,C_SCHED_DAY];
   HOLIDAY       = TT[I,C_HOLIDAY];
    TOT_WORKED    = HOURS_WORKED + UNS_WORKED;
    IF (NOT SCHEDULED_DAY) THEN
       { /* Not a normally scheduled workdaY */
       CHARGED_OT    = TOT_WORKED * 1.5;
      IF (HOLIDAY) THEN
          {CHARGED_OT=MAX((TOT_WORKED-8),0)*2.5+MIN(8,TOT_WORKED)*1;}
    }
  ELSE
      { /* Is a normally scheduled workday */
         CHARGED_OT = MAX(TOT_WORKED-NORM_HOURS,0)*1.5;
       IF (HOLIDAY) THEN
            CHARGED_OT = MAX(TOT_WORKED-NORM_HOURS,0)*2.5;
     }
    TT[I,C_OT] = CHARGED_OT;
END;
```

Figure 2: RS/1 RPL code fragment for overtime hours calculation

A pleasant surprise was how well a decision tree handles an on-line help system. By reflecting the system's menu structure and task flow into the structure of the decision tree it was possible to access it from the 'trunk' as is usual, and also to jump into the middle when called from a menu in the middle of the application.

The help system was also structured so that the explanatory text was kept in a separate table and referenced from within the decision tree. Options were also provided to print the contents of the table containing text, thus making the system somewhat 'self-documenting'.

Maintenance

The only maintenance required has been when the plant has changed its rules and procedures around calling out and charging for overtime. Careful task partitioning in the system design resulted in these changes being easy to make, with little to no side effects throughout the system.

Benefits

With the system in use in twelve areas on the plant scheduling approximately 350 workers, a savings of approximately $30,000 per week could be attributed to the system; determined by the following calculation: Prior to installing the system the overtime pay as a percentage of total pay was 18.7% (of 14,390 hours/week = 2691). After using the system for 10 months it was down to 3.8% (of 16,289 hours/week = 619, a reduction of about 2000 hours). Although other factors were affecting the level of overtime, supervisors directly attributed about half of the reduction to the system (1000 hours @ $30 per = $30,000).

Factors contributing:

- Increased awareness of 'where the time was going'. Simply seeing reports of the overtime summarized by the reason and skill worked caused many supervisors to re-examine their practices and re-position manpower to minimize re-occurring needs.

- Decreased number of mistakes. By keeping close track of all the shifts to be worked scheduling unnecessary or duplicate manpower was avoided.

- Ability to schedule work ahead of time rather that call-out at a moment's notice. Scheduling just one shift ahead of time, rather than waiting until the last minute can save $50.

Other benefits not directly measured in the bottom line also accrued. The workers came to trust that the system was treating everyone fairly; grievances filed with the union for overtime reasons dropped to 10% of the original number. Schedulers found that they could perform their jobs better, under less stress, because the system was handling the detailed paperwork, automating the simple parts of their jobs, and freeing them to apply themselves to more challenging tasks.

Lessons Learned

The story of this system has taught us new lessons and reinforced old ones:

- The methods of knowledge engineering used can have as large an impact as any other decision in the system design. An expert system succeeds or fails based upon the form and quality of knowledge extracted from the experts.

- Involve the eventual users early in the development of the system. Find out what the like or dislike; watch the way they work and use the system.

- Keep an eye out for innovative uses if standard methods.

- Employ iterative development. This will keep you from going off too far in the wrong direction.

- Design the system for flexibility and expandability.

- Know when to stop prototyping and start fielding.

The payback of the OOSS application matches what we usually see in successful systems. After one year of use the system has resulted in a bottom line savings of ten times the money 'spent' internally on its development. The hope is now that the system, with minor modifications, can repeat its success at other sites in the company.

Thanks To LaPorte

The successful development and delivery would not have been possible without the cooperation we received from the workers at the LaPorte plant. Their help in the design, knowledge engineering, and debugging made it possible to deliver this application. Thanks also to those in systems positions who helped in the integration into the existing applications and those in management who saw the value of this effort.