ASAP - An Advisory System for Automated Procurement

Robert Chalmers, Robin Pape, Robert Rieger, and William Shirado

Lockheed Palo Alto Research Laboratories
Dept 96-50, Bldg 251, 3251 Hanover Street
Palo Alto, CA 94306

Abstract

When our division changed its method of requesting the purchase of materials and services from a paper form to electronic mail, we incorporated an expert system to look for errors before the user's fingers had left the keyboard. With patient explanations, gentle prodding, and occasional subtle threats, the system has reduced the error rate from 75% to absolutely zero for the 12,000 documents generated in the first two years of its use.

The Problem

Within the Research Laboratory at Lockheed Missiles & Space Co, purchases of materials and services had until recently been initiated by the end user filling out a procurement request form (PR). Formerly, buying specialists working from little more than verbal requests used these forms to record all the data necessary to generate a purchase order. Then, as a cost saving measure, filling in the form became the requester's responsibility. The abbreviations were mysterious, the jargon was arcane, and there were 104 blanks on a single page. In the hands of scientists and engineers with a low tolerance for filling out forms and no interest in learning about procurement, this PR form was an invitation to disaster.

As a result, only about one quarter of all PRs could be processed as submitted. About one half were cleaned up by a procurement specialist either with a telephone conversation with the requester or from his personal knowledge of what the requester intended. The remaining quarter were returned with instructions for revision.

In 1989, top management began a program of automating laboratory administrative tasks based on use of the laboratory's VAX/VMS computer system. The plans for online generation, transmission, and approval of PRs were well under way when we learned of it. By restricting the input screen to the relevant items, this approach would eliminate some of the paper form's confusion, but from lack of interest or understanding, many of the same mistakes would still be made.

Our Solution

On learning of this plan, the authors proposed that most of the mistakes could be caught by an expert system module which would interact with the requester, advise him of inconsistencies or rule violations, and suggest proper alternatives. If it explained its reasoning well enough, fewer of the mistakes would be repeated. That procurement specialist with all the personal knowledge (who, it turned out, would soon be retiring) would be our human expert. The anticipated paybacks of the system were savings of time of both requesters and procurement specialists, speedier deliveries, and elimination of the clutter of faulty documents in the system.

The proposal for an expert system, later named ASAP, was accepted and it was then developed in parallel with the database program. Although the combination is known to users as the FASTBUY system, in this paper when there is no ambiguity FASTBUY will also frequently refer just to the database portion of the program which calls ASAP.

Description

Overview

The environment of ASAP is the VAX/VMS/COBOL/RDB/DECFORMS database application program which calls ASAP after the user has "submitted" a PR with at least the required minimum data that the COBOL program has accepted. A single pointer to a table of all the arguments is passed to ASAP.

When ASAP is initialized the user is advised that his PR is being checked. Within a few seconds, depending on the current VAX load, he or she is either advised that the PR has been approved or a dialogue of one or more queries and/or advisory messages is begun. Queries either call for a selection from an answer menu or free text entries which become part of the PR. Reasons are always given with queries or advice messages, but if a PR passes all of ASAP's tests, nothing but approval is indicated.

If ASAP recommends any changes, it summarizes them on a final screen which offers the user options for proceeding, then returns control to FASTBUY.

Structure

Inferencing in ASAP is performed by Neuron Data's NEXPERT *OBJECT*, Version 2. The knowledge base is partitioned in a fashion dictated by Lockheed's historical method of handling multiple related PRs, which allowed as many as nine additional items to be purchased as add-

ons to a base PR as long as they were for the same contract and vendor. A decision was made to retain this feature in FASTBUY, which forced a partition of ASAP into a base PR section and an add-on section. The control of the looping is handled in C code.

Additional functionality provided by custom C routines includes interfacing with the user, the COBOL program, and the VAX/VMS operating system for files ASAP reads or writes at run time; "reading" the free text fields; and generating automatic MAIL messages to operators when a file access failure prevents normal execution of ASAP.

The knowledge base was developed on a Macintosh, then copied to the VAX and linked with both the C code and the database program. Nexpert was chosen for the portability of the knowledge base between these platforms, its broad capability, and the developers' familiarity with it.

Because many users would have only a VT100 terminal, its use was assumed in designing the interface screens. PC or Macintosh owners use them in VT100 emulation modes. Each screen to be displayed is stored as a VAX text file.

User Orientation

Considerable attention was given to user friendliness in the design of ASAP. It was recognized that many non-computer users would feel rebellious at having this system forced upon them, and all reasonable efforts had to be made to forestall any avoidable complaints. While we only had control over our own user interface, we conferred extensively with our database colleagues to try to produce a unified appearance.

An important design guideline we adopted was to assume that the user could possibly know more about some detail of policy, procedure, or practice than ASAP. When a procedural change occurs, some user might happen to know about it before the program can be updated. Also there had been some exemptions authorized for some specific users. Therefore, the user was universally given the option of ignoring any "advice" offered by ASAP. But in case the user is wrong, ASAP then adds a message field to the PR alerting the buyer to the suspected error. The options for the user if ASAP is not entirely satisfied with the PR at the completion of the interaction are to:

- return to the data entry routine to make a correction
- repeat the ASAP analysis to change his answer to a query
- submit the PR as is
- store the PR for future revision, or
- discard it.

Avoids Repetitions

ASAP tries not to be boring. When a PR is revised and resubmitted whether immediately or later the same day, ASAP does not repeat advisory messages. For each new call to ASAP, it checks first to see if the new PR number is the same as the last PR examined. If so, only changes to data or answers to queries will retrigger rules previously fired. After each PR review that ends with the requester saving the PR instead of submitting it, Nexpert automatically also saves the state of the consultation (its logic variables) by writing them to a scratch file and naming it with the PR number. Such a scratch file is always looked for and loaded if found whenever a PR is submitted enabling ASAP to pick up where it left off. We trust our users' memories for a few hours, but not overnight. Thus, at the first execution of ASAP each new day, all scratch files are deleted, so reminders are repeated once per day.

"Reading" Text Fields

A significant custom feature we created for ASAP with C routines gives it a rudimentary text recognition capability which may be novel in some aspect of its implementation. The need first arose when our human expert informed us that a common but important error in many PRs was mischarging of overhead items such as shop supplies directly to accounts of contracts under which funds for overhead items were separately provided to the company overhead pool. Such purchases should instead be charged to the requesting department's share of that overhead pool. The practice by requesters may be either accidental or deliberate when a department has not saved a sufficient amount of its overhead money to last to the end of the year. It is important that such a practice be caught and corrected in all cases because the government rightly views this as double billing. People have been fired for this. Thus there was an apparent need to be able to infer certain attributes of a requested item that could be associated with terms appearing in the free text fields. Because this need did not require parsing of the text, which was probably unparsable anyhow, a simple dictionary system was employed.

The entries in the dictionary are those terms the requester might use in the "Specification," "Description," and "Requester Remarks" text fields which for one reason or another we wished to be able to recognize (such as "hydrogen" or "gloves") because they implied some attribute we needed to know (such as "hazardous" or "overhead") for which a strength of belief value could be assigned (range 0 to 100).

The analysis of a text field consists of seeking a match of each word in the text against a tree of all dictionary entries. In the tree, each node corresponds to a character in an entry and can contain pointers to all possible following characters. For example, in the root node the third pointer points to all subtrees for entries that start with the letter C. The maximum depth of the tree is thus only the number of characters in the longest term in the dictionary, but the breadth can be enormous. Each node must be capable of holding pointers to all 40 allowed characters: alphanumerics plus a few additional symbols including a space, making it possible to store phrases as entries.

In its compiled form, a complete dictionary entry consists of the vertical thread of pointers proceeding downward from the top node spelling out the entry term. The last letter of an entry also has a flag indicating a "hit" and a position-

ordered list of values associated with that entry for all attributes. The analysis of the text requires two tests for each source text character: for a "hit" and for an "end" which is implied by a null next character pointer. However, a "hit" does not automatically guarantee the end of a thread; both "disk" and "diskette" might need to be included. Reaching an "end" without a "hit" means the requester's word or phrase is not in the dictionary. Upon finding a "hit" the stored list of attribute values is accumulated. Then, if it is not also an "end", tracing the thread continues to the next node. At an "end" the lookup routine returns all accumulated scores and steps to the next word in the text field.

Interesting things can happen if the dictionary contains the words "computer" and "paper" and the item to be purchased is "computer paper." By themselves, "computer" implies "fixed-asset" and "paper" implies "overhead." Attempting to use different values to resolve this can only prove to be a blind alley. This can be easily resolved by adding the phrase "computer paper" to the dictionary with an "overhead" value provided that the method of accumulation of scores generates a result greater than either of the individual scores. We chose to use the EMYCIN arithmetic to do this as it also guarantees that the result will always remain bounded by 100 regardless of the number of values combined. It further suggests the possible use of negative attribute values, but so far they have not been required. An auxiliary test program showing the dictionary output values for arbitrary typed inputs was found helpful in making decisions on values to be assigned to new dictionary entries.

The dictionary is compiled from an ASCII text file with one line per entry of the following sort:

ARGON# gas = 99 *

486 PC# computer = 50 fixed-asset = 50 *

This file has been maintained both as a simple text file and as a database table, which affords some convenience for sorting and editing. Compilation builds the tree from each entry adding nodes as necessary and inserting one pointer into a node for each character preceding the # sign. At the end of the thread for 486 PC, the set of scores is all zeros except for two 50's stored in the positions corresponding to the attributes "computer" and "fixed asset."

This structure affords remarkably rapid lookup (proportional to the length of the term being sought, independent of the size of the dictionary), but is expensive in its use of memory, although only one copy is required no matter how many simultaneous users are executing ASAP. Maintenance of the dictionary is easy. A new entry is made simply by inserting a line anywhere in the source file. Adding a new attribute requires adding one line to the file defining the list of attributes and relinking the programs that compile and use the dictionary. Development of the dictionary and the knowledge base go hand in hand as new capabilities are added to the program.

This simple "reading" technique has proved quite successful for our needs, and incidentally suggests to users that the program is a lot smarter than it really is.

Development

The environment that made ASAP possible was a significant factor in the project's success. By 1988, VT100 terminal access to the laboratory's own VAX/VMS system had only recently become available to all 1000 employees. Interactive access to the main plant's administrative computing facilities ten miles away was only beginning to reach the laboratory. PCs and Macintoshes were increasingly used, but had only reached perhaps 50% of technical personnel's desks and less for nontechnical staff. Following the successful introduction of an on-line labor-reporting VAX program for department and program managers, the assistant laboratory director envisioned an integrated set of similar high level tools for all laboratory administrative functions. Automating purchase requesting and approving became the second objective.

The recent investment in a corporate AI Center within the laboratory for AI research, development, and training reflected a broad recognition of the potential of AI technology. The desirability of an expert system in the on-line generation of PRs was further impelled by the imminent retirement of the man who had led the growth of the lab's procurement organization for a number of years. Without an expert system, the error problem would get far worse after his departure. But his long experience and willingness to participate made him an excellent resource person for building an expert system. In addition, the manager of the procurement organization was enthusiastic in his support of both an expert system and the total goals of FASTBUY. ASAP was a proposal that couldn't fail. The team which proposed this application was made up of five classmates graduated from that AI training program only six months earlier and one of our instructors.

Unfortunately, the head of the FASTBUY database team was, from our viewpoint, overly cautious during the ASAP proposal phase. The design of the database program was already fairly far along, and as designer, he bore total responsibility for the integrity of the data. This, he felt, made it necessary to prohibit the expert system from writing anything but annotations to the data, and he insisted that a single call be made to ASAP so that in the event of its total failure, it could most easily be bypassed. This precluded the use of rules to check data as they were being entered which would have resulted in greater ease of use by the requester. However, we hasten to add that uniformly excellent cooperation was received throughout the execution of the project from all the FASTBUY team members.

The funding to build ASAP supported only a little more than a two-man level of effort which was generally divided equally between the knowledge engineering and C code development. Throughout the program the knowledge engineering has been the exclusive province of Robin Pape. The C coding was by the other three authors at various times.

Knowledge Engineering

Interviewing the principal expert produced most of the rule content of the knowledge base. Leafing through a large number of faulty PRs gave additional insights. Sorting through the free text fields from 30,000 PRs gave a feel for how users described items. The knowledge acquisition was the responsibility of R. Pape who also wrote and tested all the rules and developed all the user interface screens. He employed tape recording during the interview phase and was accompanied by one other person so that sessions could be reviewed and discussed for confirmation of what was learned.

The knowledge base contains about 500 rules and is basically backward chained. It is partitioned into two parts in order to accommodate the practice of data sharing when ordering multiple items for the same contract and from the same vendor (add-ons). One set of rules relates to the common information and executes only once; the other, called as often as necessary, uses the variable information. The C code handles the calls to the knowledge processes.

Examples of Rules

If the requested item is made by another Lockheed company ASAP advises the user how it can be obtained more easily by a different procedure.

If the request is for hypodermic syringes or needles, he is informed of a host of special regulations concerning "controlled substances."

If the object is a cylinder of gas and it is being charged to a government contract, several screens of dialogue ensue which require the user to promise to keep a log of the consumption of the gas as required by law. His reply becomes a part of the purchase request by means of the findings which ASAP adds to the PR file.

A rule specifying a lower limit on the procurement time allowed by the requestor's required delivery date resulted from asking procurement management what problems were found by audits of the department. Government auditors had noticed a large fraction of procurements were being demanded in only a few days. They correctly felt that either too much expensive overtime work or inadequate comparison shopping was being done. ASAP now advises the requestor that either more time must be allowed or conditions for rationed emergency procurements must be met.

Support code

The FASTBUY/ASAP application resides on one of the laboratory's cluster of VAXes that execute under the VMS operating system. It is built as a single executable file composed of several groups of objects linked together and compiled principally from COBOL and C source files. FASTBUY employs the DECForms proprietary interface building toolkit and makes multiple database calls to RDB. ASAP invokes a VMS version of the Unix "Curses" screen I/O routines in addition to calls into and out of Nexpert Object.

The interface between FASTBUY and ASAP is a single function call as mentioned earlier. The FASTBUY portion of the application is responsible for the entire PR data entry form. Once the form is filled in and the user has chosen to submit the data, it is automatically sent to ASAP for review. The data set is passed as an argument to ASAP in the form of a fixed field ASCII string, but data structures are not shared between FASTBUY and ASAP. The string is parsed to yield the PR field entries, which are submitted to the knowledge base before inferencing is begun. The ASAP support routines are written in C and use the Nexpert applications programming interface to access the knowledge base and to set up up the linkage to the routines that Nexpert itself will invoke.

During the course of a review session the expert system can ask questions and present tutorial or explanatory text. Questions can be multiple choice or a more general question with a free text reply. All of the text for display and for questions is kept in individual ASCII files. The knowledge base rules contain filenames but not the actual text. This single level of indirection was chosen to allow one to alter the text using a simple editor rather than requiring access to (and knowledge of) the Nexpert development environment.

Three Curses routines are used to generate the ASAP user interface: one to display a text file, one to ask a multiple choice question, and one to ask a question that will accept a free text reply. The routines are called from within the rules in the KB. The multiple choice replies can simply be asserted in the KB while the free text replies are analyzed and then those results asserted into the KB.

The "text reading" routines were initially written and tested on a UNIX system but when they were ported to VMS they suffered a dramatic increase in execution time. The VMS operating system supports a wider variety of file organizations than UNIX, but its implementation of random access within a file imposed a heavy penalty in overhead. VMS supports a file access technique that allows an entire file to be mapped into memory and then accessed as a simple character array. This technique was used to restore the performance to acceptable margins.

The add-ons to a PR to purchase additional items that share a common vendor and contract mentioned earlier are handled in the C code by calling multiple inferencing sessions, one for the common data that remain constant, and additional calls for each specific item's set of variable data.

The task of coding all the supporting C functions was comparable in size to the knowledge engineering task and was done chiefly by R. Rieger. The only remarkable point is the development of the "reading" capability by R. Rieger and W. Shirado described above. After the retirement of R. Rieger in August 1991, the C code tasks were carried out by R. Chalmers through January 1993.

Deployment

An incomplete prototype of FASTBUY/ASAP was first exercised by real users beginning in September 1990. Both programs were continuously expanded and reworked for the next twelve months. Production work for about 10% of the laboratory started in September 1991, about 2 years after work on ASAP was begun. The number of users was rapidly increased until the full laboratory was on-line by the end of 1991. However, development did not stop at that time, but gradually turned into maintenance as it became more user driven.

In July 1992, FASTBUY/ASAP was demonstrated under the auspices of Neuron Data, the creators of NEXPERT, at the Tenth National Conference on AI in San Jose, CA.

Through the end of 1993, about 12,000 PRs had been processed by FASTBUY/ASAP which, including add-ons, represented approximately 20,000 items procured. Not a single faulty PR had been found by procurement personnel.

The only known failure of ASAP was on March 10, 1992, when a VAX system operator took a disk off-line without realizing it held files used by ASAP. In consequence, accessibility tests of ASAP files were added to the program with automatic generation of warning messages when needed.

Costs/Payoff

The labor cost to develop ASAP from acceptance of the proposal to full production use in December 1991 was about $200,000. Two VAX runtime versions of NEXPERT were bought for the application for under $15,000.

By June 1992, it was estimated that the FASTBUY/ASAP system would save over $1,800,000 annually, and by measurement it had cut the average author-to-buyer time from 30 days to 4. There was no determination made as to what fraction of these achievements was specifically due to the ASAP program, but in general, FASTBUY gained most of the speed by substituting electronic mail for paper delivery, and ASAP probably gained the majority of the dollar savings through elimination of the effort formerly spent cleaning up faulty PRs. There is probably an exaggeration in the above total savings claim, but the savings from ASAP alone probably is somewhere between $600,000 and $800,000.

Maintenance

As with all such projects, it was not easy to determine when the system was complete. Both ASAP and FASTBUY developers had many improvements they wanted to add. Also, there was a constant stream of requests for improvements from users. Therefore, a user committee was formed in 1992, consisting of six individuals who were among the heaviest users of FASTBUY. The group was charged with determining the priority of requested and proposed improvements. Enhancements to ASAP continued to be made by the authors throughout 1992 at an additional cost of about $84,000.

Maintenance of the knowledge base due to policy or rule changes was required only once, and that occurred before the system went into production.

Subsequent Developments

A natural consequence of the successful deployment of the FASTBUY/ASAP system in the R&D Division was the decision to extend the use of the technology to the larger task of company-wide procurement. A task force to reengineer procurement was formed and consultants hired. One year later in early 1993, the ASAP team was asked to port the existing ASAP program to the UNIX-based client/server architecture of the new system. Because the larger program would supplant FASTBUY in the laboratory, maintenance of ASAP stopped and the team went to work on the port. At this point W. Shirado took over all the C and UNIX coding responsibilities.

The new architecture calls for ASAP to be partitioned with all the user interface functions executing in the requester's PC or Macintosh computer. This led to the user interface being completely redone with Neuron Data's Open Interface tools.

A server machine is called whenever Nexpert's services are required. (The commercial client/server configuration of NEXPERT is not used.) However, the responding server may be any one of several such machines. Therefore, all of a PR's data are maintained in the user's computer until the ASAP review process is completed. Each call constitutes a complete transaction even though only a small piece of ASAP is executed. Once again, NEXPERT's state saving capability is employed. Each call to NEXPERT transmits to the server the most recently saved state of the consultation, all the relevant PR data, and any new user input since the last call to NEXPERT. The server proceeds with the analysis until the next need to return control to the user. A new state file is then written and transmitted to the user machine together with instructions for any new user display.

The change from VMS to UNIX was also complex due to the many services ASAP obtains from its operating system such as automatic generation of mail messages.

Final Assessment

The FASTBUY/ASAP system has clearly been a success. The procurement people it was built to help like it. Many users of FASTBUY might still prefer to write their PRs on paper, but all users like the shorter procurement time of FASTBUY.

Nobody basically likes having his work checked, especially by a computer, but ASAP has helped users to understand the procurement process better. They like no longer having to participate in the old forms procedure that they never understood and somehow found demeaning. We think ASAP has made them more tolerant customers of the pro-

curement organization. And in a time of retrenchment, the savings from ASAP may have helped save ten or twenty of their jobs. Finally, the decision to continue ASAP into the next generation procurement program is the most gratifying result of all.

Acknowledgments

We gratefully acknowledge the contributions of the following colleagues: our classmate Abe Tassa who helped write the proposal; Drs. Linda Cook and Dan Drew, two friends who were both our instructors and advisors; Dr. Stan Salisbury, whose managerial expertise kept the skids greased and the wheels in motion; and the members of the FASTBUY development team, Dr. Ken Siler, Nancy Dixon, Bobbie Riedel, Mike Wright, and Ming Yang, whose friendly cooperation always made the work a pleasure. Most of all we are grateful to Dr. Joseph Reagan, Vice President and General Manager of the Research & Development Division who immediately saw the potential value of the project, and then entrusted it to our hands.