The CANASTA Experience: Key Management and Technical Decisions in a Hybrid Expert System Project

Anil Rewari and Mark W. Swartwout

Digital Equipment Corporation
Marlboro, MA 01752

Michael S. Register

Digital Equipment Corporation
Colorado Springs, CO 80920

Abstract

This paper presents a case study of a successful intelligent system developed for use internally at Digital's customer support centers. CANASTA1 (The Crash Analysis Troubleshooting Assistant) is a Digital proprietary knowledge based system designed to assist support engineers in isolating the underlying causes for operating system crashes, whether they be due to hardware faults or system software bugs. CANASTA's success is largely due to a combination of both project management actions, as well as the innovative technical design and development of the system. In this paper we present lessons learned from developing a large hybrid system such as CANASTA. We strongly believe that these lessons can be used as guidelines for maximizing the potential of other expert systems.

1 Introduction

CANASTA (The Crash Analysis Troubleshooting Assistant) is a Digital proprietary knowledge based system developed at Digital Equipment Corporation. It provides guidance to computer support engineers at Digital's Customer Support Centers (CSCs) in the difficult and time consuming task of analyzing operating system crashes, traditionally one of the most complex types of problems reported by customers. CANASTA has turned out to be very successful - a version that assists in the analysis of VMS operating system crashes is currently deployed at over 20 countries and is used to resolve hundreds of crash related customer calls every week.

CANASTA's success is largely due to a combination of both managerial decisions and actions, as well as the innovative technical design and development of the system. Key management decisions that contributed to the success of the project include the initial decision to select one of the most complex domains in servicing computers, thereby increasing the potential payoff; to initially convince the management at the support centers to take the risk of trying out early prototypes; to keep the management at the support centers informed, and to constantly request feedback from them as well as the expert level users. And finally, to make sure that the development team made incremental releases of CANASTA as new functions and knowledge were added on a periodic basis.

Innovation in the design of the CANASTA architecture is also largely responsible for the success. CANASTA has integrated different problem solving modules that model the different types of problem resolution strategies that experts use in this domain. These include making quick checks (rule based) on whether the crash is due to a known bug; deeper analysis (decision tree based) reasoning for resolving new types of crash problems; and checking for similarities among unresolved cases (a form of case based generalization) that can lead to the identification of new bugs in hardware or software. CANASTA also integrates AI and other technologies that have not been combined before in this domain. It integrates a remote scripting package and rule based inference for sophisticated automatic data collection from the customer's machine thousands of miles away. It uses a rule based system for quick checks on known problems. It uses a tool that allows experts to quickly encode troubleshooting knowledge graphically in the form of decision trees. It uses database technology to store case related information which can be accessed later. CANASTA also includes an innovative distributed knowledge maintenance system that automatically collects knowledge from experts worldwide at all of the CSCs, validates and redistributes this knowledge automatically to all other sites.

1CANASTA is a trademark of Digital Equipment Corporation
In the next two sections we present an overview of the domain, the functions, and architecture of CANASTA. We then highlight the lessons that we have learned from the CANASTA experience, and which we believe can lead to maximizing the potential of hybrid systems.

2 Architecture and functions

When an operating system detects an internal error so severe that normal operations cannot continue, it crashes. For many operating systems, this involves signaling a fatal condition and shutting itself down in an orderly fashion by saving the contents of the registers, stacks, and memory at the time of the crash into a crash dump file. The underlying cause for the error may be a failure in user-written code, hardware failure, or an error in system software. Crash analysis is resolving the problem, whether it be due to hardware or software, and identifying a fix or a workaround.

This is not an easy task since there is no fixed algorithmic method to identify the reason for the crash. Experience plays a large role in the problem identification process. CANASTA, by virtue of its architecture and interface to users, may be regarded as a computer assisted methodology for addressing such problems consistently.

In figure 1, the horizontal line at the top separates the customer machine that crashed from the host machine at the CSC where the support engineer works. The customer machine can be thousands of miles away. The crash analysis process begins with the establishment of a connection from the CSC to the remote machine through a modem line.

The data collection module extracts data from the crash dump file without transferring the entire dump to the CSC. The information gathering activities in this module are handled by a remote scripting package and governed by a rule based controller which has available to it a knowledge base (over 750 OPS5 rules) of methods for extracting the relevant symptoms from the dump file for different types of crashes. This package allows individual commands to be run at the remote machines, and returns the output back to the data collection module at the CSC.

Once the initial symptoms are collected, then the first of CANASTA's analysis modules is invoked. The symptom-solution module uses a knowledge base of symptom-solution rules to see if the given crash matches a known hardware or software bug. If a match is found, then the solution is displayed. Rule-based pattern matching and heuristic identification of problems give considerable leverage in trying to quickly determine the cause of the problem, especially in a domain where almost half of the problems seen are repeated problems that were previously seen by others.

If a match is not found in the symptom-solution knowledge base, then the deeper analysis module is invoked. This module is organized as a hierarchical cluster of decision trees, containing knowledge about how experts troubleshoot crashes, to guide the user in analyzing the crash dump. It suggests the most appropriate tests to be performed given the particular crash and will eventually indict a specific hardware component or narrow down the set of possible software failures.

Given that there are many cases that remain unresolved, CANASTA incorporates a mechanism for collecting and characterizing these unresolved cases. CANASTA's unresolved crash processor periodically collects unresolved crash cases from CSCs worldwide that run CANASTA and, using heuristic knowledge, groups similar unresolved crashes into similarity sets. It attempts a high-level classification of the cause for each similarity set. The main beneficiaries of these sets are the experts to whom unresolved crash cases
are escalated. Now, instead of looking at a single case in isolation, they can look at that case along with a set of similar unresolved cases. Having multiple instances of the same problem type results in quicker resolutions due to generalizations that they can make. The unresolved crash processor is crucial in expediting the process of experts generating new knowledge about crash-causing bugs.

All cases seen by the support engineers are saved in a case database. CANASTA's case management module allows users to browse through both the resolved and unresolved cases seen at their site. CANASTA saves the status of a case before exiting, including information about whether the crash was resolved using information from outside CANASTA's knowledge bases. Experts can then retrieve all cases resolved by non-CANASTA means and fill the holes in the knowledge bases. Furthermore, administrative and technical statistics can be obtained from the case database.

Finally, maintenance of the knowledge bases is essential to CANASTA. New bugs in hardware and software are continuously identified, and previously identified bugs frequently extend to new releases of software or might have better solutions available at a later date.

For maintaining the symptom/solution knowledge base a tool was developed that allows experts to enter knowledge of known problems in the form of templates into an exclusive crash related textual database. The templates are basically textual renditions of the rules in the knowledge base. The CANASTA template consists of essential symptoms that identify a problem, and textual attributes such as description of the problem, technique for confirming the problem, and the solution. The tool dynamically checks whether the values being entered are valid. It also checks for consistency among different symptom values (there are constraints in the relationships between some of the different symptoms). The maintenance tool resides at CANASTA sites worldwide so that crash analysis experts from various locations can enter knowledge into CANASTA. Once the template is filled out, the maintenance tool automatically forwards it to a central site where they are parsed and translated into rules, and are then compared for consistency with all the existing rules in the knowledge base. A new rule is added only if it is found to be consistent with all the other rules in the knowledge base. The updated knowledge base is then copied over Digital's internal network by daemon processes running at CANASTA sites across the world. The knowledge base is updated and distributed on a weekly basis.

The deeper analysis knowledge base is maintained using DECtree, an internal tool developed by Digital. DECtree provides a graphical user interface that allows one to create, or modify decision trees. It translates the decision trees into C source code that is compiled and linked to the rest of the CANASTA run-time system.

The high level architecture of CANASTA was designed early on in the project. Many of the suggestions in [3] were followed for identifying the knowledge representations schemes to use. We went through several design - implement cycles, similar to the iterative cycles mentioned in [1]. The development schedule was tied to base level releases, with each base level having increased functionality or knowledge. Refer to [2] for more technical details on CANASTA.

3 Lessons learned from the CANASTA experience

CANASTA has been an extremely successful AI project as evidenced by the large business payoff since its deployment. The use of CANASTA at the CSCs has resulted in substantial savings to Digital in handling crash related calls. Other benefits include: a convenient repository of crash related information that can be used to get statistics about various pieces of software and hardware; the use of CANASTA as a training tool for new users; a means to cope with employee turnover, especially the departure of expert level engineers; and feedback to the operating system design groups regarding problems in the operating system.

Looking back at the almost three years of involvement in the design, development and deployment of a hybrid system such as CANASTA, there are a number of important lessons that we have learned. These lessons can be used as practical guidelines for maximizing the potential of other hybrid systems. These lessons fall into two categories: project management actions that contribute towards the acceptance and success of such systems, and technical factors that play a key role in maximizing the potential of such systems.

3.1 Project management actions that maximize the potential of hybrid systems

This first set of lessons relate to issues typically handled by a project or product manager. These have to do with deciding what problem to work on, determining release schedules, and handling the myriad
of inter-organizational issues required for a successful project. Some of the decisions taken here are also required on a more traditional project, but their application in a high risk AI project may be less than obvious.

1. Project selection: Building a commercial AI system requires substantial effort and commitment. It is important, from the management perspective, and especially as part of an AI group with limited resources, to select the right problem domain so as to maximize the potential payoff from such an effort. In our case, the area of crash dump analysis proved to be an ideal domain. It is a complex problem domain requiring much experiential knowledge to quickly resolve problems. There is a large volume of such problems needing resolution on a continuous basis. There is a shortage of expert level engineers in this area and there is a high average resolution time for crash related problems at the support centers.

Applying AI solutions in such a domain maximizes the potential for high payback. The resolution time will be greatly reduced, and the quality of the solutions will be greatly increased through use of an expert system that captures rare expertise and applies it accurately and consistently. Choosing a domain where the volume is large multiplies the individual effects. This results in bigger payoffs in cost savings and increased customer satisfaction.

2. Building a foundation of support from the management and experts at the deployment sites: Our experience over the last several years in deploying CANASTA and other AI systems has been that it was critical to have the support of both the management and the experts at the deployment sites (in our case the customer service centers). The groundwork of this support needs to be built by the management and project team developing the AI system. The first step is to sell the idea of an AI solution in the problem domain. This is done by demonstrating an understanding of the business problem, and presenting the potential for high pay back if the effort succeeds.

Once the management and the experts at the deployment site become keen on the idea, then we have to maintain their interest and involvement. In the case of CANASTA, we constantly kept the management at the customer service centers informed about the state of the project and delivery schedules. We solicited feedback from them and the experts at these sites, and incorporated much of their feedback in incremental releases of the system. This last part, namely, seeing their ideas actually implemented, was important in their accepting the AI system. Of course, once the initial phase of development was over and the cost savings became real, the management at the deployment sites became strong supporters of the AI system with little further convincing from us.

3. Deploy the AI system at a large site/group of users right from the onset: The CANASTA system was deployed right from the onset at Digital's largest customer service center. This resulted in many benefits for the development team. First, because of the larger base of users, we obtained more complete feedback on the functionality and problems that needed to be addressed. Second, we were able to get performance figures when large number of users used the system concurrently. This is especially critical when using LISP or LISP based tools since the application can come to a virtual standstill when used concurrently by many engineers on the same computer system. Third, one can get more accurate statistics about the effectiveness of the AI system. Fourth, there are more experts available at the larger sites so that the knowledge base will grow faster with them being involved. Finally, the use of the new system at the largest site motivates users at the smaller sites to try it out as well.

4. Regular releases of base levels: Most AI systems do not follow the normal software engineering cycle. Instead, they follow an iterative cycle of refinement and deployment. We found that during the early stages of deployment when CANASTA did not have the entire range of functionality that it has now, the management and the users at the deployment sites expected regular releases of new versions of the system with incremental increases in functionality or performance in each release. We had sold them the idea of an intelligent and powerful system that would ultimately save them a lot of time and money, and at the same time would increase customer satisfaction. While they were realistic in not expecting all the functionality and power at once, they did expect constant progress towards reaching those goals, and nothing was a better indicator to them than the periodic release of new base levels, with each release having more functionality. The iterative cycle that is typical of AI development projects fits these expectations extremely well.

5. Transitioning from prototype to product: CANASTA was deployed at the service centers even when it was just a prototype. Given the success of the prototype, a decision was made to turn CANASTA into an internal product. However, we could not simply withdraw the prototype and come back a year or two later with the product, because the sites were us-

---

2The term "product" here means a fully documented, robust, integrated application, not necessarily something for sale.
ing the prototype in solving customer crashes. We found it appropriate to write a product development plan that involved incremental replacement of functional modules in the prototype with product quality versions. For instance, the case management module in the prototype was based on LISP code interacting with indexed files. In one of the incremental changes to the prototype this was replaced with a C based module interacting with a relational database, while other prototype modules remained the same.

We also found that going from a rapid prototyping AI effort to a product involves transitioning of development engineers as well. The type of engineers involved in the design and advanced development stage of the prototype were mainly software engineers trained in AI and knowledge acquisition. They were very successful in rapid prototyping without being bound to the rigors of the normal software engineering cycle. Once the prototype was found to be successful and the decision to make a product was taken, there was a phasing in of software engineers who were more familiar with developing products and had worked under the more rigorous software engineering cycle. They had little or no knowledge about AI. They were provided with the design and architecture of the successful prototype, and their job was to implement it efficiently and robustly. There was an overlap time during which the AI engineers brought the new engineers up to par with the design of the system after which they were phased out.

3.2 Technical factors that maximize the potential of hybrid systems

The second set of factors relate to technical design and development decisions made during the course of the project. The CANASTA architecture was designed early on, but the languages and tools that we implemented the early prototype in, and the order in which functional modules were developed and released, resulted in feedback from which we gained many lessons.

1. Automate data collection: In many domains, as in crash dump analysis, collecting the relevant information is itself a problem. Some symptoms that are critical to resolving the problem require substantial knowledge to isolate. For instance, in the case of crash analysis, certain symptoms are isolated only after tracing through the stack of procedure calls, and one has to have a good understanding of the operating system as well as the assembly language being used. Collecting the same symptoms from different versions of the operating system, or different CPU types, may require different techniques. Finally, tough problems require deeper analysis, and the knowledge required to isolate relevant symptoms becomes more specialised.

Our experience has been that automating the collection of data had a significant impact on the success of CANASTA. Users have certain expectations of AI systems, whether reasonable or unreasonable, and acceptability of the system is bound by how well those expectations are met. In our case, users expected that besides the task of problem resolution, the system would perform the mundane and often difficult task of collecting the key symptoms. This task required intelligence as well, and so they expected the AI system to help them here as well. In our first base level release of the prototype we concentrated on the problem resolution modules (where the real intelligence resides) and made available a simple interface through which the users had to manually enter the data. It was left up to the users to isolate the key symptoms. This was found unacceptable to many of the users, and many sites did not bother to use the prototype at this stage.

Besides acceptability, we found out that automatic data collection significantly enhanced the performance of the system. A rich knowledge about data collection techniques allows the AI system to send out dozens of commands to isolate certain parameters, if the need arises, and do so in seconds. It significantly decreases the time required to collect the symptoms. It can isolate certain parameters with increased accuracy. And because it knows about alternate techniques to get certain data, it can explore another technique when one technique does not return a value. This results in an increase in completeness, i.e., complete data for almost all cases. These latter two factors, increased accuracy and increased completeness, can lead to more accurate problem resolution.

2. Different problem resolution strategies in the expert system: A key factor that has contributed towards CANASTA's success is the availability of different problem resolution modules to the user. CANASTA's architecture reflects the different strategies that support engineers at customer support centers often use. Often, at customer support centers where numerous customers might call to report the same problem seen at their sites with respect to a particular product, the support engineer first tries to identify if the current problem being investigated is due to a known bug. S/he does so by scanning the textual databases and electronic bulletin boards, or by asking their colleagues. In crash analysis, almost fifty percent of crash related calls received at the support centers are due to problems that have already been identified. CANASTA's symptom/solution module captures this
strategy. Furthermore, since most known problems that cause system crashes are now routinely entered into CANASTA's symptom/solution rule base, it can find out whether the current crash is due to a known bug in a matter of seconds.

In case the support engineer is unsuccessful in finding out whether the current problem is due to a known bug, then s/he must use knowledge about the domain to analyze the problem more carefully. The more experience the support engineer has, the greater the chances of isolating the cause of the problem. CANASTA's deeper analysis module does just this—except it uses the knowledge acquired from top experts in this domain. Therefore, even if the problem was not caused by a known bug, a user can use this module in guiding him/her to analyze the problem and arrive at some indictment of a hardware fault or a software bug most of the times.

Finally, a certain percentage of cases remain unresolved at the frontline and are escalated to more senior engineers. It is of immense help to get multiple instances of the same problem, even though the instances are unresolved. By comparing multiple instances of similar looking unresolved cases, one can arrive at resolution much faster. CANASTA's unresolved crash processor module greatly facilitates this strategy.

We feel that there are numerous other domains for which problems seen at the support centers could be attacked by expert systems which have these different types of problem resolution modules.

3. Integrate conventional systems with intelligent problem solving modules: A key problem in the early stages of CANASTA's deployment was that we did not integrate high performance conventional systems with the AI based problem solving modules. At that time our focus was to provide powerful analysis modules that captured the troubleshooting techniques used by experts, since these were the modules that were essential to assist users in resolving crash related problems. Therefore, we did not provide automatic remote data collection using a conventional remote scripting package. Instead we simply presented an interface where the users had to manually enter information. We did not use a high performance database for saving the case information, but instead used LISP based code interacting with indexed files. After just a few hundred cases had been saved in this facility, the support centers found that it took minutes to retrieve cases from the indexed files. Also, using a LISP based system that was accessed by perhaps a dozen or more users concurrently became a sore issue at the support centers, mainly due to the need to increase certain system parameters, user quotas, and memory on their systems. We found that by not integrating the right conventional tools even at that early stage dissuaded some support sites from using CANASTA.

In the latter versions we systematically integrated high performance conventional tools with the problem solving modules. CANASTA now uses a state-of-the-art remote scripting package that allows it to automatically collect data thousands of miles away on the remote machine. Case information is now saved in a relational database which allows very fast access even with thousands of cases in it. Users can now generate detailed reports using their own SQL interfaces. CANASTA's knowledge base maintenance systems uses an existing high performance textual database system as a front end that allows users to enter rules in the form of textual templates.

By virtue of this integration of high performance conventional tools with the intelligent system, CANASTA is itself now regarded as a high performance problem solving system by users, as opposed to the prototype that it was in the early stages. This has contributed towards its acceptability at places where it was not readily accepted earlier.

4. Distributed knowledge maintenance: Expertise in almost any domain is hard to find. In domains like operating system crashes, by virtue of the constant introduction of new hardware and software, new bugs are being constantly discovered. In such domains it makes sense to have numerous experts participate in updating the knowledge bases. In the case of CANASTA, we have a pool of experts located at support centers worldwide who are encouraged to contribute knowledge about crashes to CANASTA's knowledge base. We have set up a distributed knowledge collection and dissemination process for this to happen. This is especially useful when one geographical area sees a particular problem before others because the product causing the problem was introduced in that geographic area first. By the time the product is introduced in other areas, problems caused by it have already been entered into the knowledge base by experts in the first geographical area.

Another useful idea is to make the knowledge maintenance tool available to functional groups involved in development as well as support of the particular product being serviced, which in our case was the operating system. The development groups release new versions of the operating system periodically, and engineers there might realize early on that certain bugs have slipped through. If they enter the knowledge about
these bugs into the knowledge base on a proactive basis even before the support centers see this problem, it can end up saving time and effort later.

5. Create and process a common pool of unresolved cases: We feel that it is important to provide some assistance to even the expert level users to whom the unresolved cases from the frontline support engineers are escalated. By collecting unresolved cases one can use AI techniques to group and classify these cases, and then present similar sets of unresolved cases to the experts. This leads to significant decrease in the time for identifying new bugs in software and hardware because they are able to make generalizations or eliminate possibilities quicker. For instance, in the case of CANASTA, our main expert estimates that over 50 rules about new bugs were generated by him alone over an 8 weeks period as a result of accessing these sets through CANASTA. As a comparison, in the previous year when this module in CANASTA was not available, it is estimated that the knowledge generated from all experts at the largest support center during the course of the entire year was less than 100 bugs (rules).

4 Summary

CANASTA represents a major success in the deployment of intelligent systems internally in Digital to assist in customer support. In leaving this project behind us, and moving onwards to work on new AI based diagnostic systems, it made sense for us to spend some time in introspection, and isolate key decisions, actions, and design ideas that contributed towards the success of CANASTA. We found that both project management related actions and technical design ideas played important roles in contributing towards the success of this system. The managerial actions fostered support and involvement from the users and their management. These actions also sustained the perception among the management at the user sites that they had a stake in the success of CANASTA. On the other hand, the technical innovations contributed largely towards maximizing the performance of users, and from the management perspective, maximizing the potential for savings. We strongly believe that the lessons learned from the CANASTA experience can be used to maximize the potential of other hybrid expert systems.

References

