

Empowering Users: A Case Study of an Innovative Approach to Resolving Conflicts between the IT Department and their Business Clients.

James Mattar, CIO for Shields Health Care Group

Peter Green, Systems Architect, BellHawk Systems Corporation

Written October 1998

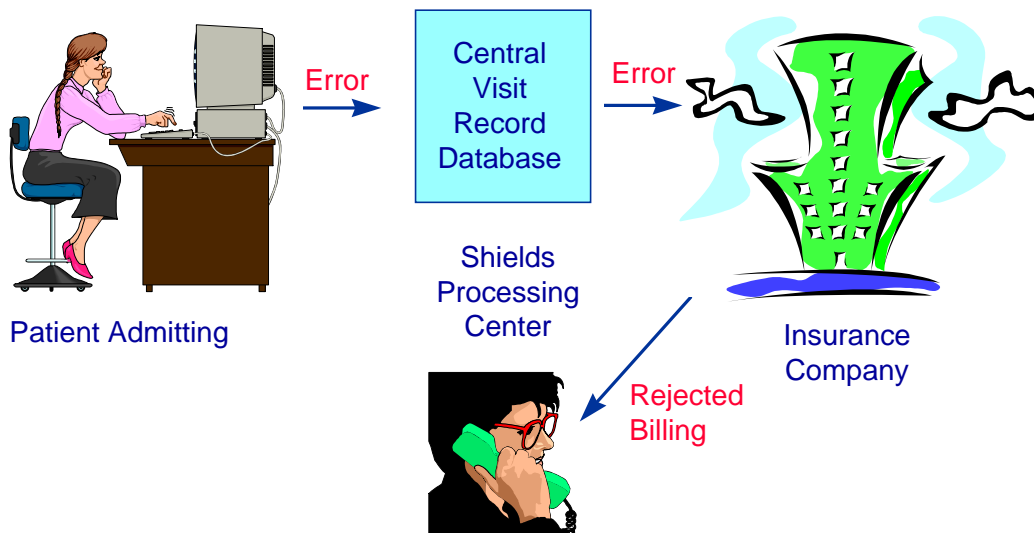
Updated February 2005

Introduction

This story has a happy ending but it started with a lot of unhappy people.

Shields Health Care operates a number of MRI Centers throughout southern New England. Patients are referred to these centers by a large number of medical doctors on an outpatient basis. Shields also operates MRI centers to serve hospital patients. The Centers process thousands of patient visits each month. Reimbursement for Shields' services comes from health insurance combined sometimes with payments from welfare and co-payments from the patients. Shields deals with approximately 12,000 health insurance companies, each with different rules for their health insurance plans.

When a patient arrives at an MRI Center they are greeted by a patient advocate. One of the advocate's primary job is to ensure that all the patient insurance data has been entered into the computer. Except in emergency, this is done before the Patient receives their MRI. This data is used to automatically generate claim forms which are submitted to primary and secondary insurers as well as welfare claims forms when needed.



Serious problems arise if the data on the claims forms are not correct. In this case the Insurance Company "bounces back" the claim. A Shield's claim specialist then has to try to resolve the problem. Inevitably this requires contacting the patient to get supplemental information before

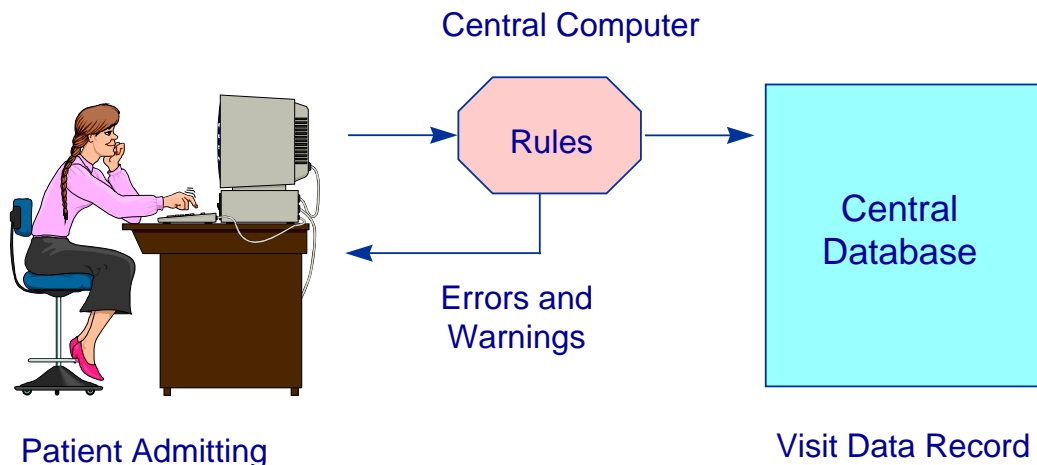
resubmitting the claim. This process can easily take half to one hour of a claims specialist's time for each bounced claim.

At the beginning of this story, Shields was experiencing a significant number of bounced claims which required a large number of people to process. This not only reduced Shields' profit from its MRI Centers but also significantly delayed payment from the Insurance Companies. It was also requiring Shields to train its admitting persons to have extensive knowledge about the intricacies of health care insurance which was resulting in escalating labor costs. As a result there was substantial pressure on the IT Department to solve this problem.

Round One

Most of the patient admitting is done using terminals connected by modems to an HP/UX® system running the Ardent UniVerse® database system to service the terminals. The interaction with each terminal is programmed in a dialect of Basic. Some terminals have been replaced with PC's running COSTAR® to emulate the terminals. The terminals are used by both the patient advocates and people entering data supplied by a Doctor's office when an appointment is made. Here we will refer to both these classes of people as admitting persons.

The first solution to this problem was to program insurance claims rules into the Basic code which assists the admitting person in collecting correct information. A typical rule decodes the ID on the health insurance card issued to the patient by looking at certain digits. This is then related to an insurance plan and used to suggest appropriate action to the admitting person. For example, many insurance plans require a patient co-payment. This may come from the patient, from secondary insurance, or welfare as the case may be. It is critical that information on secondary and tertiary insurers be collected before service is rendered otherwise Shields may be unable to collect this money. Also many plans have restrictions that may not allow the patient's insurance to cover the planned MRI. Again it is critical to determine who is going to pay for this treatment before and not after an expensive procedure is performed.



At first, this solution proved to be a major improvement as admitting persons were immediately alerted to insurance problems . As a result they were able to gather much more complete and correct information, resulting in a significant reduction in bounced claims. Soon, however, a major problem began to appear. Even with combining rules for similar insurance companies, the

number of rules quickly grew to over 250. Also insurance companies frequently make changes to the rules concerning health plan offerings. As a result, the maintenance of these rules grew to consume a major fraction of the time of one of Shields' key IT support people.

While senior management appreciated the benefits of the rules, they were unwilling to increase the IT budget to cover the cost of hiring a full time senior person to maintain the rules. As a result of a heavy workload for the IT staff, rules maintenance was largely assigned to trainees and interns who were only available for very limited time periods. This resulted in the rules becoming outdated as requested changes to the rules were sometimes taking many weeks to implement. This disenchanted the insurance claims specialists who stopped asking for changes to the rules.

The outdated rules were giving wrong advice to the admitting persons which resulted in a high level of dissatisfaction with the admitting system. As the rules became increasingly outdated, admitting persons were trained in "fudging" the advice produced by the rules by ignoring the insurance claim error warnings, thereby negating the value of the rules.

Round two

By this point in time we had a lot of unhappy people. The relationship between the IT department and the business users of this system had degenerated into a classic finger pointing exercise. The business people were blaming the IT department for being unresponsive to their needs. The IT department felt that their business users didn't understand that they were doing the very best job they could, given their budget constraints. Even worse, this problem and its lack of resolution frequently became a major topic at meetings of senior company managers which put significant pressure on Shields' CIO to find a solution.

At this point in time, Shields' considered such alternatives as eliminating the rules or limiting the rules to those that did not change frequently. Neither was acceptable as they negated the original purpose and value of the rules. Shields then turned to BellHawk Systems Corporation and funded BellHawk Systems to do a study to determine whether the rules technology that BellHawk Systems has developed for the US Air Force could help resolve this problem. As a result of this study, an innovative approach was jointly devised which empowers users to change their own rules while enabling IT to control the testing and deployment of the rules. This proposal was implemented and resulted in rave reviews for the IT department from its business users. It also resulted in a reduction of the IT maintenance time to under 15 minutes per week and, as a consequence, resulted in a much happier CIO and IT Department.

Some of the user comments after two months of operation included:

"Three Cheers for Everyone that Worked on Putting this On-Line. We think Its Terrific!!"

"Thumbs up from Mass Bay - It has worked out great for us"

"The error exception program is a big improvement over the old program as the errors now reflect current policy and "fudging" is no longer necessary. I always felt that "fudging" ended up causing more errors than if there were no rules at all."

How the Problem was Solved

BellHawk Systems Corporation is a software solution integrator that has spent many years performing research into distributed intelligent agent technology. This research was primarily funded by the US Air Force under the SBIR program. As part of this research BellHawk Systems developed an English-like declarative rules language called the Decision Support Language (DSL) and a compiler to translate these rules into C/C++. This language enables users to express the processing of their data as a set of declarative rule statements each having the form :

```
IF Condition THEN Action;
```

The benefit of using this notation is that the resultant code is easy to read and understand by non-programmers. Over the years BellHawk Systems had added many features to DSL to enhance this readability based on inputs from its user community of many large DOD contractors. This resulted in code that was easy to understand and that could be validated by subject matter experts who were not programmers.

In attempting to solve the Shields' problem, it was proposed to rewrite the rules (used to check the insurance data collected by the admitting persons) from Basic into DSL. The initial concept was to have the rules still modified and maintained by the IT department but in this new language. This was quickly ruled out because any savings reduction in maintenance time was negated by the additional burden of learning and maintaining yet another language.

It was then that we took a great conceptual leap. Suppose that we could have the insurance claims specialists maintain their own rules. This would empower the insurance claims specialists to have control over the rules used to verify the data collected by the admitting persons. Further it would enable them to incorporate rules to give advice messages to the admitting persons according to the situation. If we could achieve this, then the benefits were obvious. By incorporating appropriate rules, the insurance claims specialists could prevent the errors occurring that caused them so much extra work in handling "bounce backs" of insurance claims.

The benefits of empowering the users were obvious. But would it work ? There were about 25 specialists involved in handling "bounced claims." Nearly all of these were computer literate in that they had a PC on their desk and could use a word processor. But none had any training or knowledge of programming. Could we train these people to write and maintain the insurance claims rules ? The eventual answer was a resounding YES but we approached this task with great trepidation.

The first step was to convert a subset of the rules into DSL and to present these to the supervisors of the claims administration groups. The idea was to see if the supervisors thought that their people could handle the task of maintaining the rules. The rules naturally fell into about 12 groups and we decided to translate two groups and then present these to the supervisors.

Issues in the Conversion Process

The DSL compiler is available with two possible execution environments. In RT-Expert™, a set of rules are compiled into a C/C++ subroutine. In Activation Framework® rule sets are compiled into concurrently executing intelligent agent tasks . We chose to use the RT-Expert™ environment because it was easier to integrate into the UniVerse® database environment. We

elected to retain the existing 12 groups of rules and to map each one into an RT-Expert™ subroutine. The subroutines generated from compiling the rule sets were then to be integrated into a process that included the code to access the UniVerse® database.

In translating the rules from Basic to DSL we made use of an important capability of the DSL compiler to split a DSL Rule Code Object into two parts. The first part is the declaration of variables and procedures that can act upon these variables, together with rules that relate to the interface of the user rules to the rest of the system. The second part are the action rules themselves, in this case a set of rules relating to a particular group of health plans.

We recognized that in doing this that we were able to divide the code into two parts:

- Systems integration code whose maintenance is properly the domain of the Information Technology Department.
- The insurance claims rule sets whose maintenance is properly the domain of the business users, namely the insurance claims specialists.

It was recognized that, in making this division, that:

1. The systems integration code would change infrequently, if ever, whereas the claims rules would change weekly. This meant that the maintenance burden on the IT Department would be minimized if we could largely automate the integration of the rules into the operational system, which we did.
2. That the claims specialists would only have to learn how to code IF...THEN...ELSE... rules on predefined variables, which is a limited subset of the DSL language. They would not have to learn all those parts of DSL that are rightfully the domain of a programmer such as declaring data structures, calling subroutines, and accessing databases. This significantly reduced the problem of training the claims specialists to maintain their own rules.

In translating the insurance claims rules from Basic into DSL we quickly recognized that there were many repeated constructs that could be replaced by using the Macro preprocessor feature of the DSL compiler. Thus we started with a rule construct such as:

```
IF .... THEN
  BEGIN
    IF ERRORTYPE = WARNING THEN LogWarning(LineNumber,
      "Check that issue date of card is before July 30th 1998");
    ELSE Logerr(LineNumber, " .....
  END;
```

and reduced this to

```
IF .... THEN WARNING IS
  "Check that issue date of card is before July 30th 1998";
```

We found many examples where we could simplify the syntax by preprocessing and took advantage of all of them to reduce complexity and thereby training time for the end users. In doing this we were taking advantage of a principle that the necessary complexity of a

programming language is directly related to the complexity of the domain it expresses. By limiting the domain to insurance claims rules we were able to significantly restrict the complexity of the language needed to express the rules.

It should be noted that the resultant "insurance claims" language in which the insurance claims specialists code their rules is not a subset of DSL. Rather it is a "cousin" whose syntax, in several cases, violates the grammatical rules of DSL. Rules written in the insurance claims language are translated to DSL rules, which are integrated into their RT-Expert rules subroutine wrappers and then converted into C/C++ code.

The User Environment

Having dramatically simplified the complexity of coding the rules for the insurance claims specialists, the next problem we faced was the environment that they would use to develop and maintain their rules. We wanted to minimize training by making use of the Windows PC's on their desks. We decided that each user would develop their group of rules on their PC and then copy these to a "Test" HP/UX system. This "Test" computer is used by the IT department to convert, compile, and test the rules before they are installed on a production HP/UX System.

The first question we had to answer was what software would be used by the claims specialists to edit and create the rules. We considered developing a point and click interface that would allow users to incrementally define rules. At first this seemed straight forward until we realized that such a tool would not only have to create new rules but would have to edit existing rules. This quickly led to the realization that this tool would have to include the macro expander, parser, syntax and lexical analyzer from the DSL compiler. In fact, such a tool would operate on the abstract syntax tree and the list of data objects produced by pass two of the DSL compiler. We would then have to create a new pass three to emit rules instead of C/C++ code. As project funds were limited, and there was an urgent need to solve the business problem, we abandoned this approach.

Instead, we elected to have the insurance claims specialists use Microsoft Word to edit their rule sets. This had the advantage that they all used Word on a daily basis to do word processing. We also provided them with an extensive help file with many sample rules that they could cut and paste into Word to form new rules or constructs. This help file contained a list of all the variables that they could access and the functions that they could use to operate on these. The variables had names like "patient", "employer", and "insurer" that were translated into code to access a number of different database tables by the macro preprocessor of the DSL compiler. These macros were predefined by BellHawk Systems and are now maintained by Shields' IT Department.

We provided a number of icons on the desktop of each user:

- An icon to edit the set of rules that that the user was responsible for maintaining
- An icon to access the help file
- An icon to check the syntax of the rule file after editing
- An icon to copy the edited file to the HP/UX system for integration.

The syntax checking was performed using a Windows version of the DSL compiler.

While, as computer scientists, we felt the use of Word to be a "lesser" solution, this proved to be a great operational success. Part way through the project, we presented the supervisors of the insurance claims groups with examples of rules and demonstrated how they could use Word and the help file to edit their rule sets. We did this with great trepidation as rejection of the proposed approach would have essentially killed the project.

In a two hour meeting, we presented the concepts and demonstrated how to use Word to edit the rules. To our relief, we got acceptance and buy-in. The supervisors were delighted that someone was finally trying to solve a major problem for them. The use of Word to do the editing of the rules gave great familiarity and eliminated many perceived operational barriers. In fact most of the meeting revolved around how could they construct rules to solve operational problems that they were currently involved in handling. Considering that the supervisors had never written a line of code in their life, this proved highly gratifying.

Based on this input we proceeded to convert all the existing rule groups from Basic to DSL and expanded the Help file. We then set up a training class, which we planned for one afternoon. We felt that this was all that was necessary as by this point we had greatly simplified the process and the user rules language. In fact, we covered all the materials, and completed hands-on demonstrations and training in just over two hours. Everyone said they understood completely and wanted to go back to their own offices and work on their rules. We then had the lead BellHawk Systems developer go around to each group and set up the desktops on their PC's and explain how to use the tools again.

We expected that the groups of insurance claims specialists would need extensive "hand holding" to edit their rules, but this did not happen. We had a Shields' IT specialist and the BellHawk Systems lead developer on call, available to immediately assist the claims specialists. However, very little assistance was requested and this was easily provided by the Shields' IT person with occasional telephone support from the BellHawk Systems person.

The groups of insurance claims specialists worked very hard for about two weeks at editing the rules that were translated from the old system. Most of their time was spent on operational issues, deciding what rules should be in the system and verifying that these rules correctly reflected health plan rules. Relatively little time was spent on the mechanics of editing and syntax checking the rules, although several supervisors admitted having some frustrating times during the learning process.

Operationally we had expected each group of rules to be handled by one person. Instead each group of insurance claims specialists worked as a group on several sets of rules. Today, the supervisors of each group do the changing of the rules based on inputs from their people. They prefer to do this, as it gives them control and oversight as to what rules are in the system.

After some initial major changes to the rules, the process has settled down to where the supervisors are changing or adding an average of 5 rules a week. Sometimes there are two updates a week and other times several weeks go by without a change to the rules. The IT Department is able to quickly integrate these changes into the operational system with little effort. As a result, the insurance claims specialists are able to quickly benefit from the changes

they make to the rules. This motivates the people maintaining the rules as they are now in control of their own destiny. If they get a lot of "bounce backs" they can fix the cause by adding or changing the rules.

The System from an IT Perspective

A major concern of the It Department was how to maintain control of this process. Having the claims specialists change their own rules relieved the IT department of a major maintenance burden. If this ever resulted in a data entry system that crashed or hung, however, then this would be a major problem.

It was decided to have the claims specialist submit their rules by copying the updated rules to a specified directory on a server and then send Email to the IT person responsible for systems maintenance. The IT person then uses a "Make" file to automatically translate the sets of rules into C/C++ code for the HP/UX, compile these, and integrate them into a process. The resultant process is invoked by the Universe database program to check the terminal input from the admitting persons.

The resultant process is placed on a test server and run against a test database to produce a printout of responses to a large number of test cases. The results of this test are distributed to the claims specialists for verification. If there are problems with the rules then the rules are changed and the process repeated. Because a lot of thought goes into the rules and syntax errors are caught when the rules are entered there are rarely problems that require any alterations to the rules. In nearly three months of intensive changes to the rules there has never been more than one correction cycle required. We attribute much of this to the readability of the rules language used by the claims specialists.

The test database is maintained by the claims specialists entering cases of interest as if they were an admitting person. This enables them to enter test cases that they are trying to catch before they become "bounce backs". All entries by the admitting persons are entered into the claims database before they are checked by the rules process. Normally the process is invoked to check a specific record and pass the errors and warnings back to the person. The system also has a mode where the a set of records can be checked and the output printed along with the input record if a problem is found. Normally this is used for checking claims before they are submitted to the insurance companies (in case the persons ignored the warnings). This same feature is used to generate test output for the admitting persons covering both old and new test cases.

Once the claims specialists report that the test cases worked OK then the new rules process is substituted for the old while the system is live. This "live update" was very important as some MRI Centers can operate around the clock. The rules process is invoked for each record input by an admitting person. From a processor usage viewpoint, we could have devised a more efficient mechanism. We considered integrating the rule sets directly as subroutine into the database program. This would have required several hours of down time to rebuild the system whenever the rules changed. Invoking a separate process, however, enables us to copy a new process over the old while the system is running without interrupting operations. This has been done many times now without any problems.

From an IT viewpoint, the benefits of the resultant mechanism are as follows:

1. It ensures that the rules process has been tested so that it does not crash the system before it goes live.
2. Claims specialists get to validate the correctness of their rules. If the rules are wrong and "bounce backs" occur then it is the claims specialists problem not the fault of the IT department.
3. The total time required for each rule update is about 15 minutes of IT specialists time.
4. All the IT work can be done during regular office hours (no graveyard shift work).

Originally it was planned to create the rules checking process once a week on a scheduled basis. This was to minimize the load on the IT department. Now, however, the process is so simple that the IT Department updates the rules process whenever changes are available. This provides very rapid response to the claims specialists which further encourages them to keep their rules up to date.

One important byproduct of this new process was a significant reduction in requested changes to rules from one group of people. Originally this group made frequent requests to the IT department for rules to be added and changed that sometimes seemed to be of limited operational value. Now that they are responsible for changing the rules they have made very few changes.

Another important byproduct was the elimination of the need for the IT Staff to be educated in the nuances of insurance claims processing. When the IT Department maintained the rules they had to understand all the nuances of processing claims. This meant that they had to become experts themselves in the arcane rules of Welfare and BlueCross Blue Shield payment eligibility. Now that this is the responsibility of the claims specialists, all the IT department has to know is how to integrate the rules into the rules checking process. This means that the IT functions can be performed much more interchangeably by IT staff rather than having one person (who understands claims processing) do all the work.

Project Notes

The project lasted about 6 months from start to having the system operational on a daily basis. We tried to involve as many people as possible at each step of the way so as to get buy-in. We did not start by writing a systems specification. Rather we started with a clearly defined vision and goal and then cyclically iterated the design and implementation towards a successful conclusion. At each step of the way, we solicited and got feedback and input from people who would end up using the system. As a result it became their system not the developers' system and we achieved a very high level of "buy-in" when the system went operational.

What Did We Learn ?

1. First and foremost, that empowerment works. If the people with the problem are given the tools to solve their own problems then they will be much happier than if the problem is solved for them. "Give a man a fish and you have fed him for one day. Teach a man to fish and you have fed him for life."

2. That problems can be decomposed into a part that is rightfully the domain of the IT department and a part that is rightfully the domain of the business user. It is not necessary that the user understands the intricacies of computer technology. More importantly, it is not necessary for the IT department to understand the details and nuances of the user's business operation.
3. That using a declarative rules language, customized to the problem domain, dramatically simplifies training and enhances users' ability to specify and maintain their own business rules.
4. That using familiar tools, such as Microsoft Word, to edit the rules makes the process much more familiar and leads to easy user acceptance.
5. That collaboration works. We successfully involved as many people as possible at all stages of this project. As a result, the users "owned" their system long before it finally went operational.

Acknowledgements

This is to acknowledge the key role played by Maura Switzer who made everything come together within the Shields' IT department, Dima Brumberg of BellHawk Systems who did most of the systems development and rules translation, Aaron Laznovsky and Dan Ginsberg of BellHawk Systems who worked on simplifying the rules language, and especially Pam Horsfall, Lauren Quimby, Sally Bolster, Karen Welsh, Linda Casey, Karen Scalia, and Nancy Clement, and all the other Shield's staff members who have worked so hard to make this system a success.