Remote Service Software at the California Research Center

Stephen R. Savitzky* Jamey M. Graham*

ABSTRACT

Last year the Ricoh California Research Center transferred two different software technologies related to Remote Service: FIXIT and REST. FIXIT is an expert system for interactive trouble-shooting; it uses a Bayesian inference engine with a product-specific knowledge base, coupled to a software agent that retrieves relevant documentation automatically. REST, in contrast, is an object-oriented application framework, a software development tool that makes it easier to construct applications that communicate with remotely-located office machines. FIXIT is now in production use in several locations in the US, helping customer-service representatives solve facsimile and multi-function machine problems over the phone. It has been ported to a Japanese version, and will soon be alpha tested in Japan. REST has been transferred to the NIS Development Department, where more than 10 engineers are using it to build the next generation CSS (Customer Support System) software.

1. Introduction

Broadly speaking, "Remote Service" covers any situation in which a person (typically a Ricoh employee) is working with a customer at a different location to diagnose, repair, or configure an office machine. Today, such an interaction takes place by telephone, although we expect that in the future the Internet may be used in many such cases. In some cases the interaction may be taking place "person-to-person" through a voice connection, for example, a help desk situation; in other cases the interaction may be "machine-to-machine" as in the CSS (Customer Support System). In some cases, such as remote set-up of a facsimile machine, both kinds of communication may be involved.

Last year the Ricoh California Research Center (CRC) transferred two different technologies related to remote service. FIXIT, an expert system for interactive trouble-shooting, is a software application used to assist a customer service representative in "human-to-human" interactions.

* California Research Center, Ricoh Silicon Valley Inc.

REST is an application "framework" that makes it easier to develop software that operates in the 杜achine-to-machine" mode.

2. FIXIT

2-1 Background

In recent years help-desk environments in customer service organizations have become more and more popular because of the number of customer problems which can be resolved over the phone verses dispatching a technician into the field. This of course implies a cost savings based on a reduction in down-time experienced by the customer and the cost of sending a technician into the field. Many of today's help-desk systems are based on the either decision tree or rule-based methods which are difficult to maintain and which do not handle novel situations cleanly.

These systems also do not provide adequate access to the vast quantity of technical documentation related to the topic of the diagnosis. Instead, "help-text" is provided which is typically a reproduction of the original technical documentation, contains less information and

requires additional resources to maintain. In most situations technical documentation is essential to the troubleshooting process given that the documentation often describes the details of how to replace, repair, maintain and even troubleshoot the components of a machine.

At CRC we have developed a new decision support environment called FIXIT which uses a new kind of inference mechanism based on Bayesian belief networks. The system also provides a novel method of automatic or query-free information retrieval into a full-text database containing online technical documentation. In the next few sections we will describe the technology behind FIXIT and then how the system is being used today by operational divisions within Ricoh.

2-2 The FIXIT Solution

2-2-1 Bayesian Inference

Bayesian belief networks have been around for centuries but have only recently become a primary tool for decision support environments. The bayesian belief network (informally known as the "bayes net") is a causal network of faults and symptoms based on probabilities [1]. For instance, figure 1 presents a small bayes net to represent knowledge about troubleshooting the fault "Dirty Rollers":

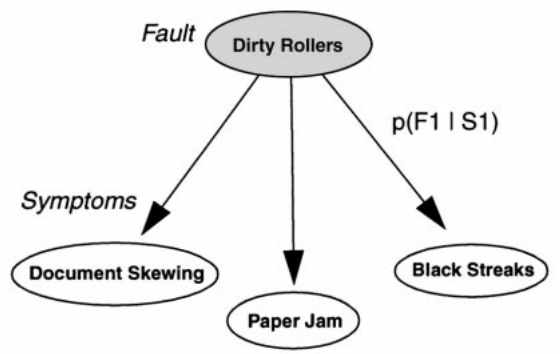


Fig.1 Simple Bayes Net

In this figure, the observation of any symptom presents the likelihood that the "Dirty Rollers" fault is present. Each arc from the fault to the symptom represents a conditional probability. For example, given symptom S1, we can compute the probability for fault F1 as: p(F1|S1). The advantage of probabilistic inference is that a probability can always be generated for a given fault no matter what the evidence suggests. This means that even when unknown situations arise during a troubleshooting session, the system will not break down as, say, a decision tree-based system would. Instead the system will produce a probability for all faults known to the system. At each step in a diagnosis, we calculate a probability for each fault which includes all evidence gathered so far, e.g., p(F1 | S1, S2, S3,..., Sn). This probability distribution is further evaluated by an activation predicate (AP) which filters out irrelevant faults which do not pertain to the current diagnosis. The AP essentially limits the number of faults to a list of 5-10 top candidates. The bayes net technology used in FIXIT was developed by a company called Knowledge Industries (KI). They provide both a graphical knowledge base development tool called DXpress and an API for embedding their technology in your own code. For the FIXIT project we created our own interface on top of the KI API as well as developing all of our own knowledge bases.

2-2-2 Automatic Information Retrieval

Access to online documentation is becoming more and more relevant in the context of accessing large volumes of information in a efficient manner. Help-desk operators often utilize technical documentation while troubleshooting a problem with a customer's machine. Many help-desk systems do not provide the adequate technical documentation online. Instead, "help-text", a summarized version of the hardcopy technical documentation, is typically used. Help-text typically does not contain the complete technical information found in the hardcopy document, requiring future updating and maintenance as product information changes. Utilizing the exact technical document found in hard copy has two distinct advantages: 1) users are familiar with the content and know how to manually search for information and 2) documents are maintained by a publication division responsible for updating the technical information. In the FIXIT system we use the electronic version of the exact hardcopy technical documentation. This documentation comes from Ricoh Corporation's Service Publication Division.

FIXIT's information retrieval mechanism is not your typical keyword based system. FIXIT provides what we call "*query-free*" information retrieval [2]. Query-free information retrieval is a paradigm in which queries are constructed autonomously and information relevant to a user is offered without explicit request. FIXIT uses an intelligent agent to monitor the users's interactions, recording the faults and symptoms which form the context of a session with a customer. The agent, working in the background, invokes an information retrieval process which gathers relevant documentation related to the current session. The user is never required to enter a query, nor must the user wait for the results of a search; FIXIT performs the search in the background and then informs the user, in a subtle way, when relevant documentation is available. The technology for FIXIT's information retrieval search engine was also created here at CRC and is based on the original table of contents (TOC) of a document and a new concept called Contextual propagation" [3]. The contextual propagation method is used within a natural language understanding system to perform comparisons based on the semantic information found both in the source (the query) and the target (the topics of the document). Because TOC topics are elliptical, information is propagated from higher level topics to subordinate topics to construct a more precise representation of the content of a section in the document.

2-3 Field Experience

With a bayesian belief network inference engine and an automatic information retrieval system running in the background, FIXIT provides a solid foundation for doing "real-world" troubleshooting, whether it be from a help-desk or running on a laptop computer in the field. In 1995 we conducted a three month alpha test of the FIXIT system at Ricoh Corporation's National Communication Center (NCC) in Lombard, Illinois. The NCC uses first-line dispatchers to answer phone calls from customers who have fax, printer and copier machines. First-line dispatchers are not trained in the skill of troubleshooting Ricoh products; they are only trained to record compliant information provided by the customer. Before FIXIT, the first-line dispatchers would simply record customer problems with a brief description. They would then send this information to a field technician who was dispatched to the site to perform the repair. Now FIXIT is used to record all complaint symptoms the customer describes. The dispatcher, lead by the embedded knowledge in FIXIT, asks the customer to perform actions on their machine and report the result in an attempt to fix the problem. If the problem is resolved, the case is closed. However, if the problem can not be resolved over the phone, the case is escalated and sent to either a technical dispatcher or a field technician who is dispatched to the customer's location.

For the alpha test we created a knowledge base for Ricoh's 3200L fax machine which, at that time, generated the most service calls at the NCC. For three months the NCC used FIXIT and by the end of the test recorded a 75% resolve rate. As a result of this improvement in customer service, the NCC made FIXIT an operational part of their regular service; over the next year they expanded to provide a FIXIT PC on every dispatcher's desktop (about 16-18 dispatchers), and built knowledge bases for approximately 35 Ricoh products. Currently they resolve between 50-55% of all calls processed by FIXIT. They have also created a fulltime position, called Knowledge Engineer, for the person responsible for creating and maintaining knowledge bases. The NCC is also quite pleased with the user interface which allows new dispatchers to rapidly learn the system reducing the amount of training time needed.

2-4 Transferring Expertise

After the success with the NCC we were requested to transfer the technology to Japan where a similar alpha test will be conducted. With the help of Mr. Tatsuo Ito of the Information Communication Research Center (ICRC) and Ricoh's Technonet subsidiary (NCC's Japanese equivalent), we are now in the process of fully testing the system in Japan. We have created a Japanese version of FIXIT (informally called J-FIXIT) and converted some of the English knowledge bases used by the NCC and expect to begin an alpha test in the Spring of 1997. If all goes well we plan on transferring the source code

to Japan where a more complete Japanese version of FIXIT will be modified and maintained.)

3. REST

3-1 Background: The Software Bottleneck

Unlike FIXIT, which is a complete application program, REST is a software tool that enables Ricoh's programmers to build new applications much more quickly than before. Moreover, the resulting applications have the potential to be more reliable and easier to modify and maintain than their predecessors. Because software is extremely complex, and because producing software is very labor-intensive, it often takes longer to design and implement the software associated with a new product than it does to develop the product itself. When the software is finished it is difficult to test, full of errors, and difficult to maintain. Nevertheless, the usual way of developing software is to start from the beginning for each program, specifying and designing a unique solution for every purpose. Sometimes it is possible to re-use modules from previous, related programs, but more often each program is developed by a different group, and there is little contact among them. For example, Ricoh has many applications for remote service, including CSS (Customer Support System) for copiers in the Japanese market, and several different "RDS" (Remote Diagnostic System) programs for Facsimile machines (with Japanese and overseas versions developed separately). Early versions of the RDS programs, especially, were completely re-written for each different product family. Each program implemented many of the same functions: communication with the remote machine, access to a database, security, information display, and user interface. But each was completely separate; not a single line of code had been re-used.

3-2 The REST Solution

In 1992 the Remote Service group at the Ricoh California Research Center had already implemented two different versions of the RDS program for facsimile machines when they learned about a new technique for developing software: object-oriented application frameworks. This new technique promised to produce software that:

* was more reliable,
* could be developed more quickly,
* could be maintained, modified, and extended more easily,
* had a design that could be reused.

On the other hand, applying this technique required learning a new programming language learning a new philosophy of programming, and a lengthy and unpredictable design and implementation period transferring what we learned to the rest of Ricoh. In other words, it involved a considerable amount of risk. It was decided that, as a research organization not directly involved in product development, CRC was in the best position to take that risk. That way, if we failed, no damage would be done.)

3-3 Programming with Objects

Object-oriented programming represents a major change in the way software is developed. Instead of breaking programs down into data structures and algorithms, and developing these separately, object-oriented programs are built using objects, which combine data structures with the operations permitted on them.) The structure and behavior of an object is defined by its class; the actual implementation of an object is hidden outside the module that defines it, and only the interface is public. This means that changes can be made to an class's implementation (for example, bugs can be fixed and features added) without necessarily affecting the parts of the program that use it.) Also, a new class can be derived from an existing one by subclassing. Only the data and operations that differ between the subclass and its superclass have to be defined; everything else is reused. Bugs that are fixed in a superclass are usually fixed in the subclasses as well.) Object-oriented programming has the potential to greatly increase the speed of software development and the reliability of the resulting programs, but many companies have had trouble making the transition. One of the most common reasons for this is an early failure caused by trying to do too much at once. We avoided this problem with REST by performing our experiments out of the way of the rest of the company, and only trying to transfer our knowledge after we had succeeded.

3-4 Capturing Design

A Framework is a re-usable design.

Unlike a simple class library, which is re-usable code, a framework contains entire "sample" applications, so that the overall structure of a typical application can be re-used. Rather than digging through the manuals trying to find the pieces from which to build a program, the programmer starts with a program that already works and modifies it.

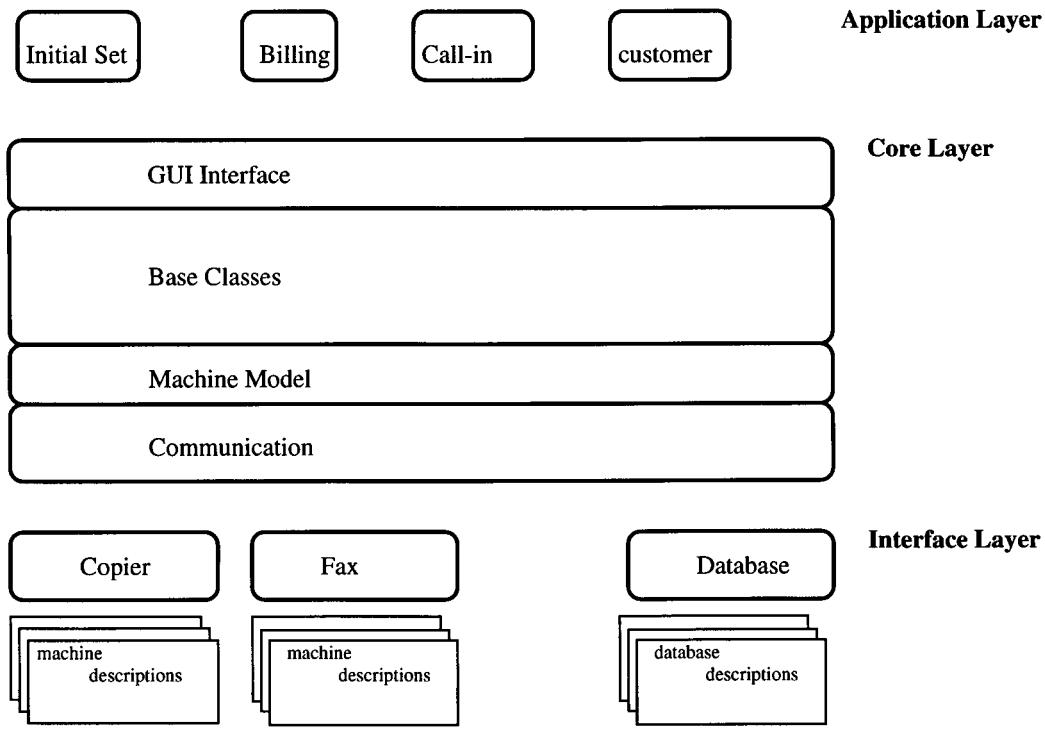
More specifically, a framework contains:

1. a library of classes that are designed to be easily subclassed (although they may be useful on their own without subclassing),
2. a collection of working sample applications designed to be easily modified and extended,
3. documentation on all aspects of the framework, including how to extend it,
4. software tools that make it easier to use the framework.

In the case of REST, we created a layered design for applications:

1. In the top layer, the application-specific code. Each application contains typically a few hundred to a few thousand lines of code.
2. a "core layer" further divided into:
   - An interface to the operating system's own user-interface framework.
   - A "kernel" of shared classes that implement collections of objects, reference-counted pointers, run-time data descriptors, and "object-oriented I/O".
   - A set of classes from which "machine models" are constructed, and that allow the application to maintain a copy of the remote machine's data.
   - Communication classes that transfer data to and from the remote machine.
3. an "interface layer" that contains device-family-specific drivers and interface code.

The "machine model" is the key to REST's effectiveness as a framework. This is a detailed description of the data structures and operations implemented on the remote machine: their data types, sizes, and locations or access methods. In effect, the machine model allows a REST application to treat the remote machine as an object, even though it was not designed using object-oriented methods. Exactly the same run-time data descriptors are used for REST's own



classes and those on the remote machine (although the remote descriptions are extended with access information and additional datatypes). A machine model is read in from a text file on disk (using object-oriented I/O) based on the type of machine being communicated with. The data descriptions in the model is then used for three purposes:

1. Communicating with the remote machine,
2. Keeping an up-to-date copy of (part of) its data,
3. Constructing a user interface.

Most of the user interface is built at run-time after reading in the machine model. This means that new machine models (i.e. new kinds of remote machine) can be added at any time without having to modify, or even recompile, any of the applications.

The REST framework includes interfaces for communicating with:

- Facsimile machines, which keep their state as data structures in non-volatile memory.
- Copiers, with state accessed through the CSS interface.
- Database servers.

Recently, another interface for communicating with Web browsers using the HTTP protocol was added; this enables a REST application to function as a Web server, using a browser as its user interface.

3-5 Results: Building a Framework

Building a framework is an iterative process.

1. The framework is designed and implemented in parallel with the first application.
2. Then, a new application is developed. This may involve changes to the framework, as problems are discovered and overcome.
3. The old applications are updated, if necessary, to stay current with the latest version of the framework.
4. The process repeats with another application.

Eventually this process “converges” and the framework becomes stable. As each new application is developed the process becomes faster. Although the first step might take twice as long as developing an equivalent stand-alone application of about the same size, each application thereafter would take much less time. When building a framework it is important to have expertise in both programming and in the problem domain. In our case, two of the three team members had previously written remote-service applications for facsimile machines. This also gave us a benchmark: by building a program with approximately the same functionality as the old one, we were able to see how the REST framework compared with the old way of doing things.

Building REST took approximately three years: one for design, one for the initial implementation, and one for refinement. This represents a longer design phase in comparison to similar projects that were not done as frameworks.

In the end, REST consisted of a core layer of about 58K lines of code, and an interface layer of about 25K lines, with applications ranging from about 500 to almost 10K lines. In contrast, a typical application (for example, CRC's RDS, a remote-setting application for FAX) might be 30-100K lines of code, none of it shared with any other application.

3-6 Tools and Documentation

Along the way, we developed several useful software tools. The most useful of these was an automatic “documentation extractor” that transforms C++ programs and header files into formatted listings and a programmer's manual, all in HTML for on-line browsing. Every mention of a class name is cross-linked to a corresponding description in the manual. An alphabetical index and hierarchical table-of-contents were also generated automatically. The automatic documentation avoids the most common problem with software projects: the unwillingness of programmers to keep documentation files up to date. Because the documentation is derived directly from the program and from short comments kept alongside the code, it is easy and natural for a programmer to change the documentation whenever the code is changed. (It is worth noting that Sun's Java language toolkit includes a similar tool, called javadoc. Our tool

appears to be better for very large projects.) All documents relating to the REST project were kept online, almost all in HTML. This meant that we could use an ordinary Web browser such as Netscape for reading the documentation. The CVS version-control system was used for source control.

3-7 Transferring Expertise

Even the best software development project is a waste of effort if nobody uses the results. All through the REST project development we had been planning its final phase: using REST to introduce Object-Oriented technology to the rest of Ricoh. Fairly early in the development process (about the end of the initial implementation phase) we invited two engineers from Ricoh's R&D facility (Imai and Katoh) to visit us at CRC. Under our supervision, they implemented the machine model and interface classes for copiers (our expertise extended only to facsimile machines at that point), and a sample (trivial) billing application. This gave us confidence that it would be possible to transfer the technology. At the end of CRC's development of REST, we held a three-month series of classes for ten engineers from RSK who would be the start of the NIS group which would take over REST in Japan, using it to implement the next-generation CSS system. There were two overlapping classes: one in application programming, and an "advanced" class in the REST internals. Each consisted of four weeks of morning lectures and hands-on afternoon laboratory sessions, followed by four weeks of working on substantial projects in small teams.

4. Future Plans

Now that the REST and FIXIT technologies have been transferred, what are we going to do next?) Our general plan is to take what we have learned about frameworks, object-oriented software, belief networks, and query-free information retrieval, and apply them to the new area of Network Office Appliances.)

Specifically, we are applying frameworks and object-oriented software to build what we call a Platform for Information Appliances (PIA).) These are networked office machines that interact with users via the World Wide Web's HTTP protocol.) Inside the appliance is a collection of software agents that operate similarly to FIXIT's information-retrieval agent -- watching the user's actions and performing useful functions in the background.) We have already built experimental agent sets for a CD-RW appliance, a Web Printer, and a digital Photo Album.)

We are also planning on using this framework and belief networks to build an advanced Web-based system which automatically annotates documents based on a user's preferences.) The belief networks will be used in this case to help recognize topics that may be of interest to the user.) Our goal is to eventually embed this technology in a future network office appliance.

Bibliography

1. D. Heckerman and M.P. Wellman, "Bayesian Networks", Comm. ACM, Vol. 38, No. 3, March 1995.
2. Hart, P.E. and J. Graham, "Query-free information retrieval", Second International Conference on Cooperative Information Systems (CoopIS), 1994.
3. J. Graham, "A natural language method of semantic pattern matching for user manual text retrieval", Ricoh California Research Center (CRC) Technical Report CRC-TR-91-16, March 28, 1991.