End-User Debugging for Electronic Commerce

Henry Lieberman and Earl Wagner
MIT Media Lab
20 Ames St
Cambridge, MA 02139 USA
{lieber, ewagner}@media.mit.edu

ABSTRACT
One of the biggest unaddressed challenges for the digital economy is what to do when electronic transactions go wrong. Currently, consumers are frustrated by interminable phone menus and long delays to problem resolution. Companies are frustrated by the high cost of providing quality customer service.

We believe that many simple problems, such as mistyped numbers or lost orders, could be easily diagnosed if users were supplied with end-user debugging tools, analogous to tools for software debugging. These tools can show the history of actions and data, and provide assistance for keeping track of and testing hypotheses.

We present Woodstein, a software agent that tracks user interaction with e-commerce Web sites in a browser, and relates the browsing events to a high-level model of complex, multi-step processes such as purchases or account transfers. Woodstein explains action steps and data in an understandable form, visualizes action history, and aids in exploration of possible hypotheses for the causes of errors.

THE PROBLEM: E-COMMERCE WHEN THINGS GO WRONG

> *"Your call is important to us…*
> *Please enter your 15-digit card number, your PIN...*
> *if you're calling from a touch-tone phone, please press 6 for customer service now,*
> *… Customer service. This is Marie speaking, can I have your card number?"*

Electronic commerce is great – you log onto a Web site, type in what you want to a search engine, fill out a simple form on the Web, give your credit card number, and Fedex brings you the stuff the next day. Simple, painless, efficient, effective.

Except when it doesn't work. People inevitably mistype numbers, misunderstand directions, or just click on the wrong buttons. Vendors lose orders, confuse names, computer systems crash. Then what?

More often than not, customers have to pick up the phone. And they probably reach a phone tree where they're forced to navigate through interminable menus, a tedious process that might or might not give the right choice of problem to be solved. It might give another phone number to be dialed. It might ask for card numbers or transaction numbers that aren't readily at hand, and have to looked up offline. If someone in customer service is successfully reached, that person (often a low-paid worker in a high-pressure call center) may specify tedious processes to be performed. They may not be empowered to actually understand or fix the problem themselves. Customers find themselves bounced endlessly from one support person to another. All of us have had these kinds of experiences.

Customer service problems are incredibly frustrating. Not only do they cause frustration about the immediate transaction, they also poison the relationship between customers and vendors. Customers feel like they are being deflected, that they are not being listened to, and that vendors do not care about their time or about their relationships. This casts a pall over future potential relationships.

Nobody expects technology to be perfect, and people are tolerant of mistakes and occasional failures, but what drives people crazy is when they get "stuck" – left with no obvious way to systematically go from having a problem to understanding what is causing the problem and how to resolve it.

But it isn't all the fault of the vendors. The problem is difficult to fix simply by companies spending more money on call centers, since human time, even if low paid, is expensive. Many companies claim that, as soon as a customer picks up the phone to call about a problem, they have already lost money on that customer's relationship.

This problem is now becoming a major obstacle to further adoption of e-commerce itself. Many people who now choose brick-and-mortar shopping over e-commerce do so simply to "have a face to talk to" in case of problems. Again, the impact of the problem-resolution experience has a disproportionate effect on the trust relationship between a customer and vendor, and nowhere is the trust issue more important than in electronic commerce.

A PROPOSED SOLUTION: E-COMMERCE DEBUGGING

Our proposed solution is provide the user with E-Commerce Debugging Tools, that enable consumers to solve many simple problems themselves from their computers. The E-Commerce Debugger is analogous to a programming language debugger used in software development. It allows the user to investigate the causes of an error by examining processes and data systematically at varying levels of detail until the problem is discovered.

Debugging is detective work – generating hypotheses, and testing them by a process of elimination. In the software development area, we have developed some novel interactive tools, described below, for supporting the investigation involved in debugging.

Of course, software debuggers are used by expert programmers and have interfaces designed to be used by expert users. For e-commerce debugging tools to be effective, they have to have user interfaces that are designed for use by non-expert users, with simple interfaces and ample explanation and visualization facilities. While we don't expect e-commerce users to understand software debuggers like experts do, we believe that most people have enough ordinary common-sense reasoning about cause-and-effect relationships to be able to use the kinds of tools we propose.

An expert programmer is confident that no matter where a bug might be, systematic investigation will almost always find the problem if a fine enough level of detail is examined. That gives them confidence in using the technology. We hope to instill that level of confidence in consumers who participate in e-commerce transactions. We also intend, where the software tools prove inadequate, to provide semi-automatic services that facilitate human-to-human communication to resolve problems.

In the design of the interfaces, we employ ideas from our experience in the intelligent software agent area, where context and learning [8] can be used to automatically infer what information is necessary, automatically find it when available, and put relevant choices at the user's fingertips, streamlining the user interface.

In the past, it would have been impossible to provide practical e-commerce debugging tools because the raw material that such tools would need – information about transactions, user identification, vendor specifications, payments, etc. would have been on paper. Now, since much of it is accessible via user accounts on Web sites, it has become practical to capture that information via Web browsers, and to provide debugging facilities that operate through the browsers.

A BROADER CONTEXT: END USER-DEBUGGING TOOLS

We actually see this work in the context of a broader goal of developing end-user debugging tools for all sorts of computer interactions, not just for electronic commerce transactions. Much lost time and frustration is experienced by computer users dealing with the intricacies of installing software, system maintenance, and problem solving with specific applications. Why can't the computer itself know what it is doing and help us solve the problems we are having with it?

Why can't we ask a computer in any situation, "What are you doing right now?", "Why are you doing that?" , "What does this error message mean?", and other questions that you might reasonably ask a knowledgeable human assistant? If something goes wrong, why can't we trace the problem back through the steps that led up to it? Why can't we formulate hypotheses about what might have been wrong, and test them until we get to the answer?

Many of the cognitive processes involved in diagnosis and repair of electronic commerce situations have analogies in many kinds of general system interaction as well. So we think our results will generalize to many other areas of computer interaction.

But electronic commerce is a good place to start, because the transactions themselves are procedurally very simple, compared to the operation of an operating system or specialized application. People understand the basic concepts of financial transactions, and since everyone participates in such transactions, interfaces to deal with them will find a wide audience.

The economic value of improving the situation is enormous, and the business community has not so far tackled the problem in a comprehensive way. Because many actions involve multiple parties, such as an online purchase involving the vendor, the consumer's credit card company and the shipping service, consumers need help from tools that operate on their behalf, rather than on behalf any particular vendor.

Further, we think the experience gained in end-user debugging could also be fed back into debugging tools for professional software developers. Since debugging accounts for approximately half of all software development costs, the potential economic payback is enormous here, too. Because development of debugging tools has so far been limited to expert professionals, we believe that insufficient attention has been paid to the usability of such tools, and we believe our usability experience could benefit these situations as well.

THE STATE OF THE ART IN CUSTOMER SERVICE

In the business world, technology is being actively deployed to improve customer service, but in fairly conventional ways. Many transactions that used to require phone conversations are now being conducted in electronic mail. E-mail has advantages and disadvantages relative to phone interactions. Sometimes it can be faster, sometimes slower. E-mail facilitates tracking and forwarding requests

and retrieving past histories. However, e-mail often does not permit real-time response to problems.

Customer support people are now supported by various kinds of databases, loosely termed Customer Relationship Management, or CRM [2]. CRM databases keep customer information and allow customer support personnel to readily access customer information. Interfaces are fairly conventional database search engines.

There are also various ways of organizing problems and solutions in databases. Frequently asked question (FAQ) lists and problem databases are often posted on Web sites and customers can look up answers themselves, as well as support people. Again, access is via fairly conventional search engines. More advanced are systems that organize problems and solutions into discrimination nets such as Answer Garden [1]. With one of these systems, the customer is led down a series of questions that eventually identify the possible source of a problem. This is often effective, but depends on fully categorizing in advance the possible problems and solutions. More advanced techniques, using artificial intelligence techniques such as case-based reasoning [13] have been tried in research contexts, but are not widespread.

All of these solutions are, of course, limited to a single vendor's server systems. Consumers encounter problems with multiple vendors and the only common feature may be the consumer him or herself. We are developing client-side agents that can have a global view of the situation.

Some sophisticated software systems provide remote diagnostics, so that a program on a vendor's machine can look for particular problems on a customer's machine and fixes can be automatically loaded. But this approach also requires that problems are identified and solutions are prepared in advance.

No present solution allows end-users to analyze the steps of procedures while they are currently engaged in them and investigate each step and data item independently as we describe.

INTRODUCING WOODSTEIN: RELATING BROWSING TO PROCESS DESCRIPTIONS

Woodstein is a software agent that visualizes actions performed by the user by interacting with web pages and helps in debugging these actions. Woodstein uses a client-side proxy to observe the interaction that the user has with a sequence of Web pages, and tries to match the pattern of interaction to *process descriptions* that describe complex, multi-step actions such as purchases and account transfers. Woodstein then provides explanation of the user's actions in terms of the instantiated process descriptions. Woodstein also permits inspection of relevant data items within a Web page so that the user can understand the role of each item in the overall history. The next section presents a typical scenario of a problem in a consumer purchase.

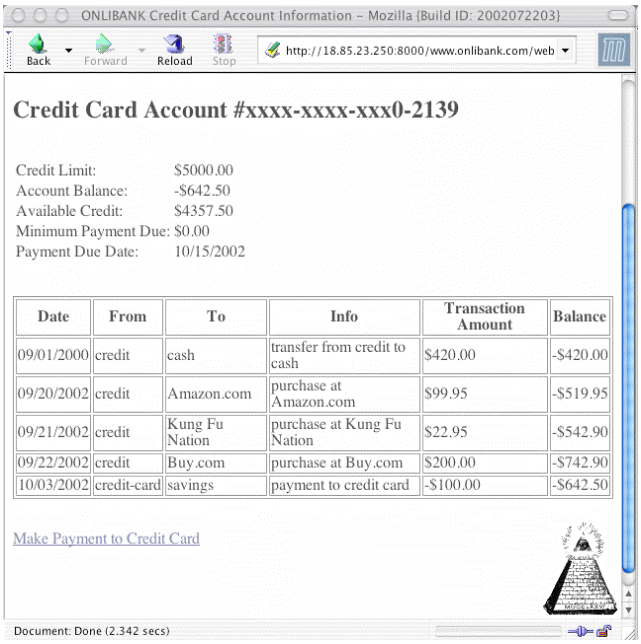

Figure 1: Viewing Credit Card Transactions

A DEBUGGING SESSION WITH WOODSTEIN

Suppose that a month ago a user bought a book from a small business on the web, *Kung Fu Nation*, for the first time, then forgot about it. In looking at the credit card statement for the previous month, the user notices that there's a charge for the purchase, even though the book hasn't arrived. The screen for the statement is shown in Figure 1.

Without Support from Tools, Debugging is Difficult

How can we understand why the book did not arrive? With current technology, there are several possible ways to investigate the situation, but all of them involve considerable work. The user may have saved the purchase confirmation to disk or printed it out. These would be accessible if the user is organized but it's more likely that the record of the purchase is lost. If the user remembers the site, and his username and password for the site, then he could log in to see a record of the purchase. However, even these details are likely to be forgotten, and records of them lost. Finally, the user could call the credit card issuer to find out more about the charge and the merchant, but this will almost certainly involve a spending a long time on the phone.

History of Online Actions

Woodstein greatly simplifies this problem. It allows the user to easily retrieve and inspect all aspects of this purchase, including the original order, the processing of the order and the delivery, without requiring explicit save and load of records of each interaction. It is able to support the

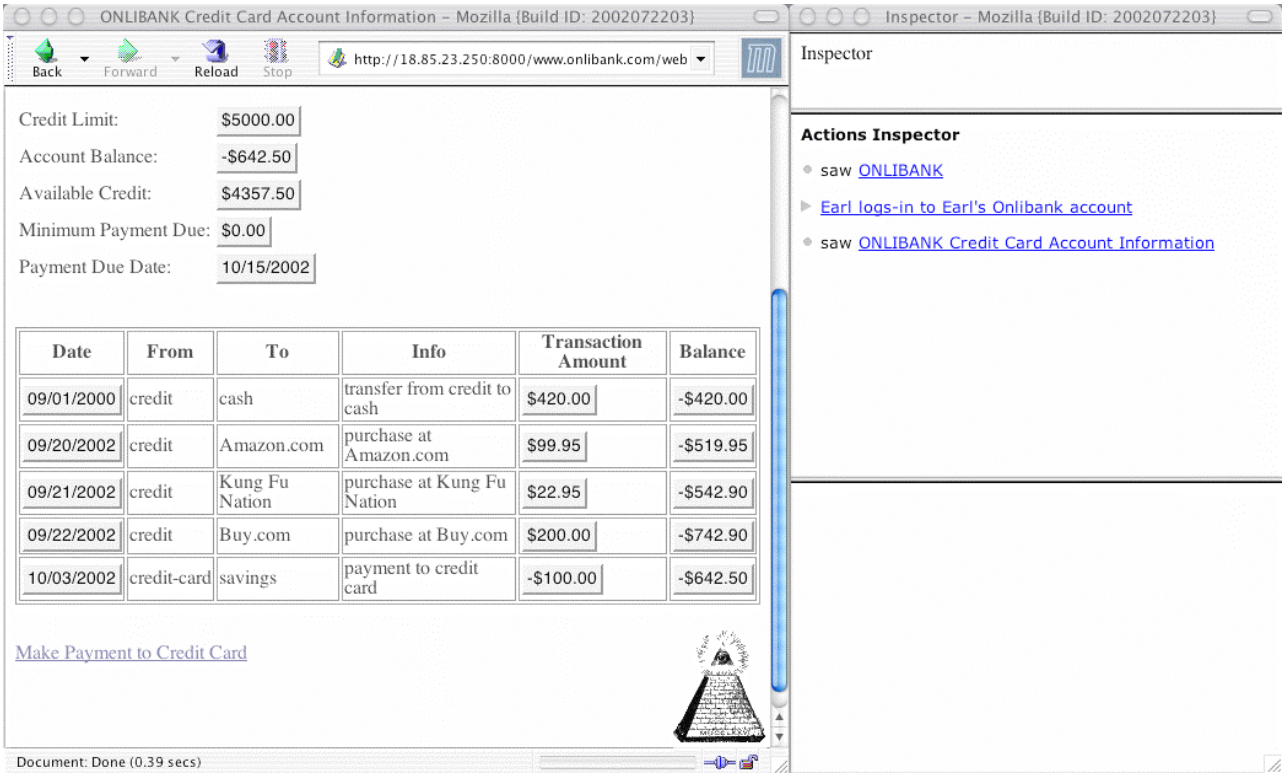

Figure 2: Inspecting Credit Card Transactions with Woodstein

user in this way because it automatically matches each Web page to process descriptions of typical Web actions. Process descriptions are analogous to program code: Actions performed by buttons on Web pages are like functions, and the data that the user fills out in forms are analogous to arguments to these functions. It allows the user to "debug" the action by examining the sequence of steps taken by the web site, related steps taken at other web sites, and the data affected by these actions. It then makes these traces accessible whenever their data appears in pages seen by the user.

Inspecting Data in Web Pages

As the user created an account and made the purchase, Woodstein recorded the pages that the user interacted with, and all of the data he or she entered. Even more importantly, it makes its records easily accessible. In monitoring pages as they are sent from the web server to the user's browser, it automatically recognizes data related to user actions. In this example, it recognized that a purchase was taking place, the amount of the charge, the date, the vendor and other information shown in the transaction history.

Ordinarily, the user simply interacts with a page by browsing or performing actions. Sometimes, however, the user would like to ask about data shown in the page and find out more about it including its history and data related to it. In this case, the user would like to inspect the purchase procedure that corresponds to the transaction shown on the page. To allow that, Woodstein supports a mode in which the user can inspect the page by clicking on a watermark, the pyramid topped by an eye in the lower right of the screen, added as part of the annotation. To find out about the purchase, the user clicks on the watermark. The first click opens Woodstein's action inspector which shows the current action.

Woodstein indicates that it recognizes that the user is currently viewing his or her credit card's transaction history. The second click on the watermark shows the data that Woodstein recognized in the current page, as shown in figure 2. Every recognized item is converted to a button that may be clicked to see Woodstein's record for the data. In this case, the user clicks on the price, $22.95, and Woodstein's inspector window updates, as shown in figure 3.

Woodstein shows its record of the purchase price in the data inspector. It gives the value for the price variable and the data structure for the transaction that the price is a part of. In this case, the user wants to interact with the transaction itself. A click on the transaction loads its record in the data inspector. The history of the transaction is accessed with another click. In the action inspector,

Woodstein replaces its record of the current action with its record of the actions that the transaction was involved in, as shown in figure 4.

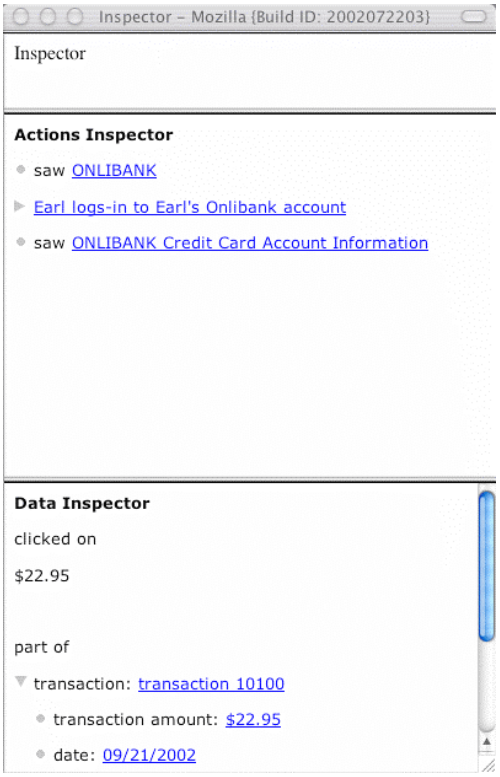

Figure 3: Inspecting Transaction Amount

User-Centered Explanations of Actions

Woodstein tries to explain actions in a form that will be understandable to the user. Although the internal representation of the action is like a function call, we generate an English-like description that instantiates the function call with the particular values that appear in this instance. Woodstein supports debugging actions by allowing the user to examine the steps that don't "look right", or which have incorrect descriptions.

Support for Generating and Testing Hypotheses

Woodstein allows the user to develop hypotheses about what went wrong, and analyze the record of the action to determine what actually happened and where the error occurred. In this particular scenario, the user may have entered data incorrectly, or the vendor may have had a problem in processing the order, or there may have been a problem with the delivery. Without Woodstein, examining these different possibilities would be annoying in the simplest case and impossible in the most complex. Though not all information is available, Woodstein goes a long way in helping keep track of information about the user's actions on the web.

In debugging this purchase, the user inspects the progress of the action clicks on the triangle beside the purchase description. This loads summaries of the steps of the purchase. The last item indicates that the book still hasn't arrived, because it reads "delivering" instead of

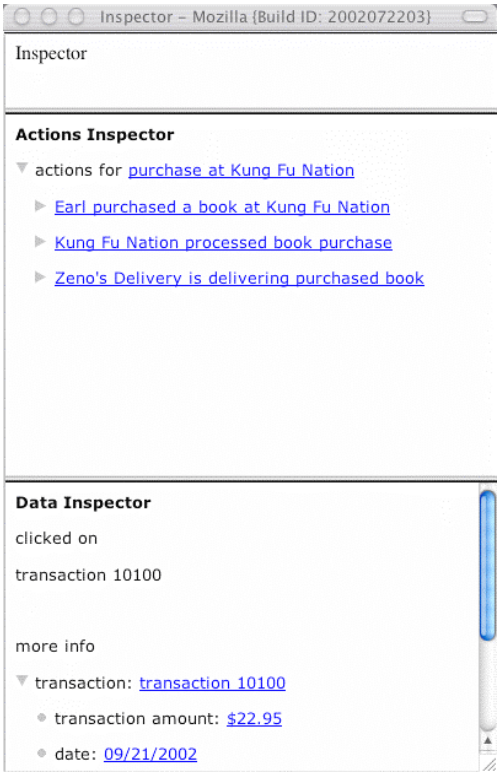

Figure 4: Inspecting Transaction

"delivered".

volved in, as shown in figure 4. Even though the user never loaded the tracking page for the purchase, Woodstein knows how to retrieve the tracking page because it's aware of the structure of the action and because it was able to identify the tracking number.. Clicking on this step loads the tracking page, showing that the item still hasn't been delivered, as shown in figure 5.

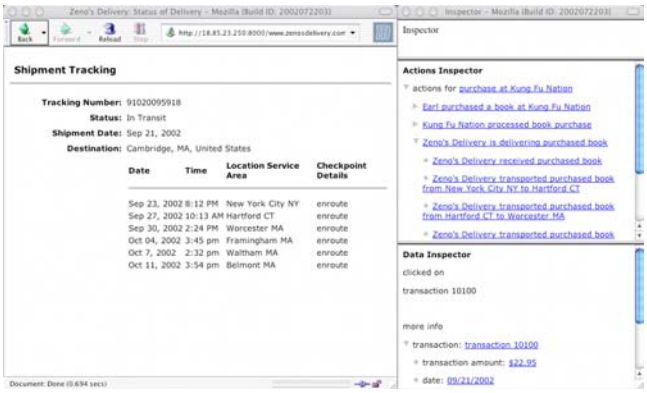

Figure 5: Tracking page

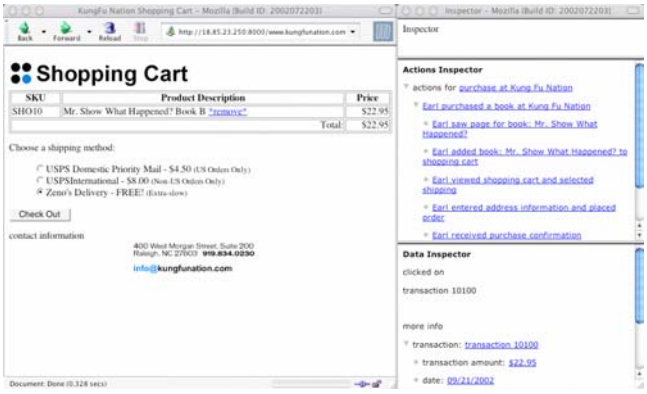

Figure 6: Shopping cart showing shipping method

Although the status of the delivery is now known, the user would like to know why the delivery is taking so long. Another click on the first step of the purchase action loads the original steps that initiated the purchase. Clicking on the step in which the shipping type is set loads Woodstein's saved copy of the page, as shown in figure 6. The user instantly sees that the "extra-slow" shipping option with Zeno's Delivery Service had been selected, explaining why the delivery is taking such a long time.

The end result is that the system can automate many of the tasks of looking up details of each transaction on myriad Web sites and correlating transactions across accounts, which otherwise would have to be performed manually. While there may be problems that can't be handled in this manner, we believe that a wide range of problems are amenable to this kind of solution, and it will result in much faster problem resolution and more satisfied customers and vendors.

TOOLS FOR MANAGING BUSINESS RELATIONSHIPS

We intend to make the user's transaction data useful in other ways, besides directly explaining to the user how a transaction has proceeded. We call these tools Business Relationship Management, by analogy with Customer Relationship Management. They could be viewed as a component of the debugging system, but they are also useful in their own right.

This data can also be put to use whenever the user is required to provide information about a transaction. For instance, if a user has to fill out a form on a web page requesting help or more information, our system will automatically identify and retrieve the relevant data and enter it as the default values in the form for the user. Not only will this ensure that the data is correct and consistent with the rest of the transaction information, the user is spared the burden of having to look up this information and enter it manually. Simple facilities in Web browsers currently help you fill out forms by making available the last entries you typed, or matching to a pre-stored form, but by tracking transactions we can provide more context-sensitive prompts and completions.

As businesses increasingly require consumers to begin interactions through the structured format of web pages, we expect that users will benefit from tools that can act as a guide in these interactions. By keeping track of the user's information and making it available at relevant times and locations, we intend to minimize the inconvenience they face.

TOOLS FOR CUSTOMER SUPPORT PEOPLE

Sometimes, it won't be possible to completely solve a problem from the user's perspective alone, because the answer might depend on details of the process or data that are internal to the vendor and that the user does not have access to. In this case, he or she must resort to communicating with a customer support person, via e-mail or phone.

But similar tools may be made available to the customer support person, so that that person can employ a similar debugging strategy. The user's agent could communicate automatically to the support person's agent, which could instantly access the details of the problem, and see the investigation the user has done thus far. Valuable phone support time would not be wasted reciting account numbers and retelling the story to different personnel in the case that more than one support person needs to be involved. The support person's interface could be far more detailed and specialized to the industry and to the company than the general public would be willing to tolerate. Typically, the user does not want to need to know the details of the company's internal process, but such knowledge might be essential to solving the problem. The customer support person could again benefit by breaking down the problem into steps, verifying each independently, tracking the history of individual data items, and so on.

DEBUGGING IN SOFTWARE DEVELOPMENT

Debugging is the dirty little secret of computer science. Despite the fact that studies of programmers show that half

of the costs of software development are consumed in debugging activities, there is precious little work in computer science that explicitly seeks to make the debugging process easier and more effective. Tools available in today's commercial programming environments are essentially unchanged from those that appeared in programming environments thirty years ago: function trace, breakpoints, line-by-line stepping.

Some of our past work has been aimed at understanding the cognitive processes involved in debugging and building explicit environments that provide operations that directly answer the questions that people need to know to debug a running program [5]. In contrast to the roles of current tools, debugging from the perspective of the programmer consists of the following cognitive processes:

- *Visualization*: What's happening? How do I get an overall feel for what's going on?
- *Localization*: Where is the problem? What specific step or piece of data is responsible?
- *Instrumentation*: Trace or "audit" the history of a particular event or piece of data.
- *Repair*: What are the consequences of a proposed fix?

We think this breakdown between current tools and the parts of debugging described above is largely a user interface problem [4]. We will use many of the principles that we have learned in constructing debugging environments for software when developing our e-commerce debugging environments, while taking into account the essential differences between the e-commerce domain and the programming domain.

ZStep [6] is one of a series of debuggers we have built that aim to support these cognitive processes, especially visualization and localization. ZStep is a reversible debugger – it records every step of the execution of a program and lets you run the program forward and backwards.

THE ZSTEP REVERSIBLE SOFTWARE DEBUGGING ENVIRONMENT

Some of ZStep's innovations include:

- An animated view of program execution, using the code editor
- A window that displays values which follows the stepper's focus
- An incrementally-generated complete history of program execution and output
- Controls to run the program in forward and reverse directions
- Incremental control of the level of detail displayed
- One-click access from graphical objects to the code that drew them
- One-click access from expressions in the code to their values and graphical output

Many analogues of ZStep's features appear in Woodstein, for example,

- Animated views of execution of e-commerce actions
- Display of action data correlated with focus in the action history
- Incrementally-generated history of action execution
- Controls to run the action history in forward and reverse directions
- Incremental control of the level of detail displayed
- One-click access from action values to the action history that produced them
- One-click access from action history to the corresponding action data

Again, we stress that, although we are inspired by approaches for software debugging, we are very conscious that we are developing tools for non-expert users, and pay very close attention to usability and accessibility for the general public.

EVALUATION

There are two ways in which we would like to evaluate Woodstein. First, an important question about Woodstein is how successful we will be in recognizing that an interaction with specific Web pages matches Woodstein's task models, in cases where human judgment would recognize a match. Further, can we match specific data items within these pages to the arguments required by these models?

To test this, we examined how successful Woodstein is in recognizing data items in pages corresponding to user action data it is already aware of. In preliminary tests with a selection of typical Web pages, it recognized 62% of the data items. We found that data items in an idiosyncratic format, such as account numbers, and phone numbers and addresses were most successfully recognized, while details such as quantities were not. While this score does not show high overall reliability for the recognition, it did result in usable models for many of our test scenarios, and left the user no worse off than standard browsers in the case that transactions had to be debugged. To improve the recognition, we also intend to incorporate a user training facility that will allow it to easily recognize many of the missed cases after a small amount of training. For more details on the evaluation, see the companion paper [10].

A more comprehensive study would test Woodstein's entire debugging interface with users against more conventional tools for representative debugging tasks. We are currently collaborating with customer service departments at two of our sponsors, Mastercard and Intel, collecting typical debugging scenarios. [Note to reviewers: We hope to have some results from tests available by press time.]

RELATED WORK

No existing system that we are aware of directly attacks the problem of end user debugging of electronic commerce transactions or Web interactions. We have talked above about antecedents of this work in software debugging and in customer relationship management. We are also working with Mark Klein [3] of the MIT Sloan School to bring to bear existing work on business process descriptions and on characterizing exceptions in business processes.

A companion paper [10] presents the tracking interface and process visualization in more depth, and provides more details concerning the evaluation.

The closest work to this in interface style and spirit is Collagen [11]. Collagen tracks user activities in a desktop application (for example, an airline reservation system) and matches these activities to plan recognition and discourse models. Collagen is like Woodstein in that the agent is capable of recognizing commonly occuring high-level goals by the concrete steps that the user is carrying out. Collagen is more proactive, and the agent is represented as a social actor in the interface and can take action directly when authorized by the user. Such an approach might also be possible for Woodstein. Woodstein provides visualization and history browsing facilities that Collagen does not, and Woodstein is explicitly focused on the debugging task, while Collagen is more conversational, and intended for normal operation rather than errors. However, a synthesis of the two approaches might be fruitful.

REFERENCES

1. Ackermann, M. and D. MacDonald, Answer Garden 2; Merging Organizational Memory with Collaborative Help, ACM Conference on Computer-Supported Collaborative Work, 1996.
2. Dyche, J., The CRM Handbook: A Business Guide to Customer Relationship Management, Addison-Wesley, Reading, MA, 2001.
3. Klein, M., A Knowledge-Based Approach to Handling Exceptions in Workflow Systems. Journal of Computer-Supported Collaborative Work. Special Issue on Adaptive Workflow Systems. Vol 9, No 3/4, August 2000
4. Lieberman, H., Introduction and Guest Editor, Special Issue on The Debugging Scandal, Communications of the ACM, April 1997.
5. Lieberman, H., Ungar, D., Fry, C. Debugging and the Experience of Immediacy, Communications of the ACM, April 1997.
6. Lieberman, H., Fry, C. ZStep 95: A Reversible, Animated, Source Code Stepper, in Software Visualization: Programming as a Multimedia Experience, John Stasko, John Domingue, Marc Brown, and Blaine Price, eds., MIT Press, Cambridge, MA, 1997.
7. Lieberman, H., Integrating User Interface Agents with Conventional Applications, Knowledge Based Systems Journal, Vol. 11, No. 1, 1998, pp. 15-24.
8. Lieberman, H., Selker, T. Out of Context: Computer Systems that Learn About, and Adapt to, Context, IBM Systems Journal, Vol 39, Nos 3&4, pp.617-631, 2000.
9. Lieberman, H. and Fry, C. Will Software Ever Work?, Communications of the ACM, March 2001.
10. Lieberman, H. and E. Wagner, Doing Stuff on the Web: Tracking the Structure of Browsing Actions, submitted to CHI 2003.
11. Rich, C., and Sidner, C. 1998. COLLAGEN: A Collaboration Manager for Software Interface Agents. User Modeling and User. Adapted Interaction 8(3-4): 315-350.
12. Riloff, E. and W. Lehnert, Information Extraction as a Basis for High-Precision Text Classification ACM Transactions on Information Systems, Vol 12. No. 3, pp. 296-333, 1994.
13. Watson, I., Applying Case Based Reasoning: Techniques for Enterprise Systems, Morgan Kaufmann, San Francisco, 1997.