Supporting User Hypotheses in Problem Diagnosis on the Web and Elsewhere

Earl J. Wagner
ewagner@media.mit.edu

Henry Lieberman
lieber@media.mit.edu

MIT Media Laboratory
20 Ames St
Cambridge MA 02139

ABSTRACT

People are performing increasingly complicated actions on the web, such as automated purchases involving multiple sites. Things often go wrong, however, and it can be difficult to diagnose a problem in a complex process. Information must be integrated from multiple sites before relations among processes and data can be visualized and understood. Once the source of a problem has been diagnosed, it can be tedious to explain the process of diagnosis to others, and difficult to review the steps later.

We present a web interface agent, Woodstein, that monitors user actions on the web and retrieves related information to assemble an integrated view of an action. It manages user hypotheses during problem diagnosis by capturing users' judgments of the correctness of data and processes. These hypotheses can be shared with others, including customer service representatives, or accessed later. We will see this feature in the context of diagnosing problems on the web, and discuss its broader applicability to system interfaces in general.

General Terms

Human Factors

Keywords

Interface Agents, Web Interfaces, Interactive Visualization

Categories and Subject Descriptors

H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia - User issues

1. INTRODUCTION

The state of consumer interfaces for e-commerce is at a crossroads. Most Americans have access to the web and many are familiar with the convenience of late-night banking and the thrill of last-minute bidding. On the horizon is the advent of web services, currently being explored and developed for business-to-business interactions, but inevitably to directly shape consumers' interfaces for purchases and online transactions. And yet most work in e-commerce interfaces for consumers—both in research and in the development of web sites—has been focused on what happens leading up to a transaction: how to provide appropriate and compelling recommendations, how often and by what means to provide promotions and updates, and so on. Interfaces for supporting customers after a transaction have received little attention and promising new trends including "customer self-service" are based on surprisingly old-fashioned technology such as site-specific searching[7].

The issue of what happens when something goes wrong in e-commerce is particularly relevant to vendors because of the risk of losing customers. A recent study found that after a bad customer service experience, 80% of customers would be less likely to buy from the online vendor again[4]. And customers are contacting support—nearly half (44%) of online buyers have made a support contact in the past six months[3]. Vendors will work to reduce these numbers, both to build the value of their brands and limit the cost of customer service. But the need for customer service will remain, just as our recently upgraded software with great new features will always have a few bugs.

Although the reasons for contacting customer service vary, some issues are more common than others. Just over a quarter of all contacts are due to a delayed delivery of a product or service and another quarter arise because of a billing or pricing issue[3]. Anyone who has made more than a few purchases online is familiar with these types of problems. Looking at your credit card transactions history, you wonder about that one order you never received. Or maybe you purchased something then discovered that you received less, or paid more, than you expected.

Experiences like these make consumers wary when initiating purchases in the first place. Indeed, the more complex a product or service is, the greater the quality of customer service will influence the decision[3], with financial services and airline and hotel reservations being the transactions of most concern. Importantly, these are also the transactions that are likely to span multiple sites, with the potential for greater complications. Consider when there's a miscommunication among multiple sites over the course of a transaction. Then it becomes the user's problem to resolve. The

IUI'04, January 13–16, 2004, Madeira, Funchal, Portugal.

user has to go back and compare each step of what each vendor did with what it *should* have done. The user must assemble a complete history of the entire transaction, then return and both seek amends from the party at fault and resolve things with everyone else.

We see an opportunity for software agents, working on the web, to provide advanced help to consumers in diagnosing and resolving their own problems, truly fulfilling the potential of customer self-service. An agent can retrieve information about an online transaction and visualize its entire history, even across multiple sites. For example, it can show the flow of payments or items as they pass from one organization to another. As we will see, an agent can even support the diagnosis of the source of a problem, by helping the user in generating a hypothesis of the cause. All of this enables users to "debug" the steps that a vendor took, that they took or even just their own mental models of a process.

More importantly, we see this technique for annotating objects with user judgments as being more broadly applicable. This approach can help with current problems in e-commerce transactions, but also with user actions on the web in general. Further, these annotations could even be applied to the interfaces of other software systems, enabling end-users to diagnose complications in the software they use.

In the rest of this paper we will discuss a software agent, Woodstein[1] with these features. Woodstein monitors user actions on the web and retrieves related information to assemble an integrated view of a transaction. We will see its "data-history" view, in which it presents the history of a transaction from the perspective of the transaction objects such as prices and quantities of stock. We will also see the agent work at an even more abstract level, in capturing user judgements of the correctness of objects involved in the transaction while diagnosing problems. The set of judgements make up a hypothesis as to the cause of the problem, and we will see how they can be shared as well as just saved for later reference. We will finish by sketching out the broader possibilities suggested by user annotations for problem diagnosis.

2. OVERVIEW OF WOODSTEIN

Woodstein is a software agent that works with a user's web browser to answer questions like "How did that data get that value?" "Why did that happen?" and "What's happening now?". It monitors a user's actions on the web to create a record of the overall process. For example, by watching the user browse an online retailer and add items to a shopping cart, it recognizes that the user is making a purchase. Later, when the user loads another page involved in process, Woodstein annotates the data items in the page enabling them to be inspected directly. Within the user's credit card transactions history page, a single charge can be examined. The history of the overall purchase process can be retrieved and reviewed, making it convenient to see the context of the charge and find out, for instance, what item was purchased.

Woodstein works by matching a user's actions to the steps of an abstract model of the process. Through this recognition, it knows to look for more information about the process on other web pages and web sites, even if the user never visited them. By watching the user select a credit card and shipper for a purchase, Woodstein knows to go to the sites of the bank and shipper to gather more information about the status of the purchase, including whether it has been paid for and delivered.

Woodstein collects and presents information about a user's data items and processes. A data item can be *simple* such as prices, addresses or dates, or it can be *composite* such as an entire transaction record or order. A process is either a user action, such as loading a page or clicking a link, or a web site action such as the creation of a new order. Woodstein is then able to explain the context and history of processes and data described in pages, such as how the items appeared in the shopping cart page. It answers questions about the history and current status of the overall process, as well as how data in the process was set.

Successfully diagnosing a problem requires an understanding of the causal relations within a system. Woodstein provides a data-history view to show the history of how a data item was computed and created. The user can revisit previous pages in its history within this automatically generated audit trail. For a purchase, the user can jump from the charge amount in the credit card transactions page to a saved copy of the order confirmation page in which the purchase price appears.

When the user feels that a process or data item looks incorrect, this judgment can be recorded through the object's context-sensitive menu. Woodstein then helps the user in diagnosing the source of the problem, even if it is just the user's incorrect understanding of the process—which is why we speak of an object "looking" incorrect. Through the process of elimination, it makes further annotations and then suggests other objects to examine. These annotations are *effective*. They are managed and extended by the agent, resulting in a record of the user's hypothesis regarding the source of the problem, as well as all of the intermediate judgements made along the way.

We will now look at an example of Woodstein's support. After, we will discuss how Woodstein works in greater detail.

3. AN EXAMPLE OF HOW WOODSTEIN SUPPORTS PROBLEM DIAGNOSIS

In this example we will see Woodstein's support for visualizing a user's actions and managing the user's hypotheses. Consider an employee of Yoyodyne who is enrolled in its stock purchase plan. Each pay period Yoyodyne sets aside a portion of his pay and, once a year, uses this money to purchase a block of Yoyodyne shares from its broker, SN-AFU. He has set up his account at SN-AFU to automatically transfer the shares to his broker, Sellwell.

The employee is browsing on the web and decides to review Yoyodyne's share purchases at SN-AFU. He notices that the number of shares most recently purchased seems lower than usual at 250, rather than around 400 (Figure 1).

The employee wants to interact with the shares directly to see their history. When analyzing this page, Woodstein added its logo overlaying the bottom right of the page. The employee turns on Woodstein's inspector by clicking on the logo. Woodstein converts all of the objects it recognized in the page to buttons (Figure 2). When the user moves the

[1]Named after Bob Woodward and Carl Bernstein, the Washington Post reporters who uncovered the Watergate scandal. When their editor, Ben Bradlee, wanted to know what they had discovered, he'd stand at the door of his office and yell "Woodstein!" into the newsroom to call them in[2].

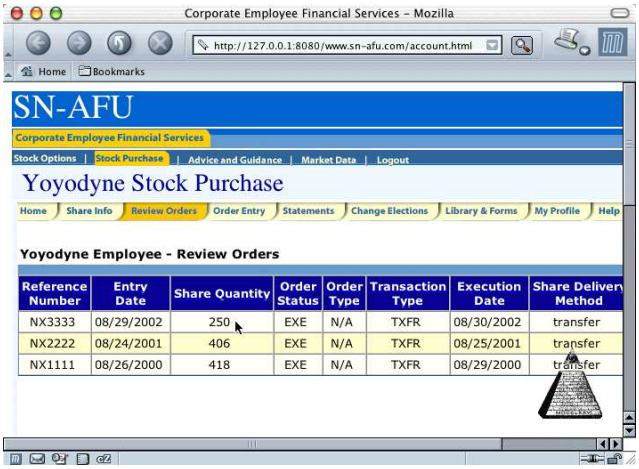

Figure 1: Viewing a stock purchase transaction

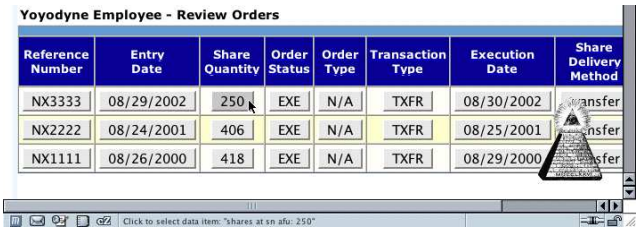

Figure 2: Inspecting the purchased stock

mouse into a button, Woodstein darkens it and updates the browser window's status bar to indicate what will clicking it will do. This explanation also shows the actual name of the button's process or data item.

The employee wants to know how the number of shares purchased was set and presses down on its button, revealing a menu. He selects "How was this set?" (Figure 3).

Woodstein opens a pop-up window showing the history of the shares with English descriptions of the processes and data involved (Figure 4). This window is the data-history view and it shows how the data item "shares at SN-AFU" was set. The top frame, in grey, answers the employee's question by explaining that the shares resulted from the process "SN-AFU set shares at SN-AFU". Each data item is set as the result of a process. Processes, in turn, take data items as inputs. The bottom frame shows how the process took the number of shares purchased as its input and set the number of shares at SN-AFU as its output.

Woodstein created this record by matching the Yoyodyne's original concrete steps in exercising the employee's stock

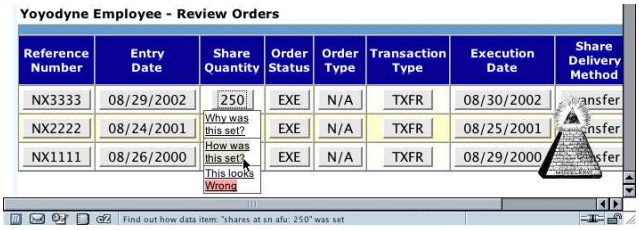

Figure 3: Asking how the stock was purchased

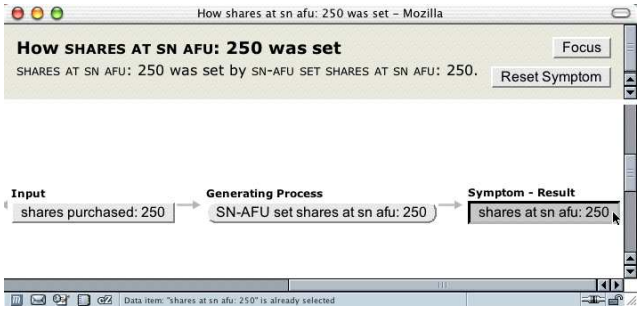

Figure 4: Viewing how the stock was purchased

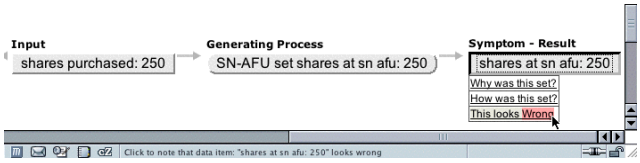

Figure 5: Noting that the quantity of stock purchased looks wrong

purchase plan with its abstract process model for the action. After Yoyodyne purchased the stock at SN-AFU, Woodstein knew to look at both web sites and "follow the money" to track the stock transaction. In this view, Woodstein presents the information it gathered from the sites involved in the purchase, revealing the first few steps back in the history of the shares.

Processes and data items are both presented as buttons in Woodstein's inspection mode, but process buttons are rounded while data buttons are rectangular[2]. All buttons for the same data item or process are equivalent across different views, so interacting with an object's button in the page is the same as interacting with it in the data-history view. Woodstein provides multiple views to show the different relations among the processes and data. For instance, a page shows the original context of objects, but the data-history view shows how they are causally related.

Returning to the scenario, the employee thinks that the "shares at SN-AFU" data item looks incorrect. He presses down on its button in the data-history view and selects "This looks Wrong" (Figure 5). The button turns red to indicate it has been annotated as looking incorrect. Woodstein then marks the objects involved in setting the data item, both the process that set it, "SN-AFU set shares at SN-AFU", and the input to the process, "shares purchased", yellow to indicate that they may be incorrect (Figure 6).

In response to the employee noting the incorrectness of the

[2]As is standard in data-flow diagrams.

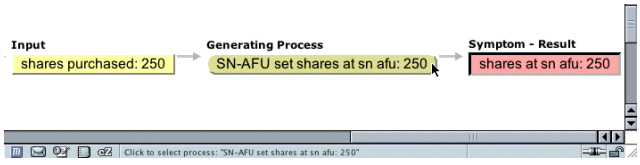

Figure 6: Viewing how the quantity of stock purchased was wrong

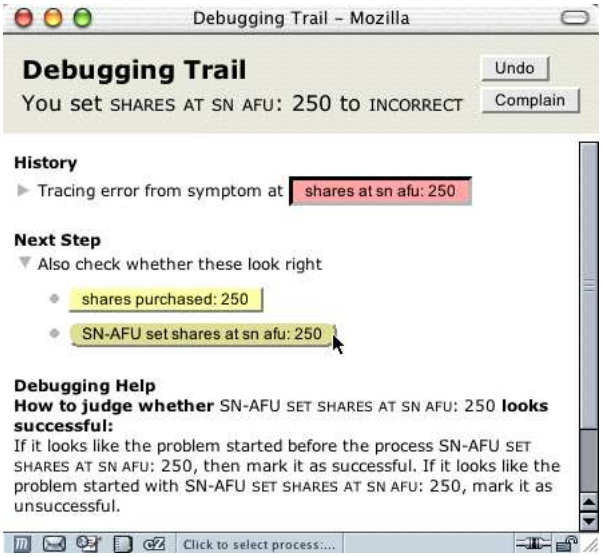

Figure 7: Viewing the record of the diagnosis

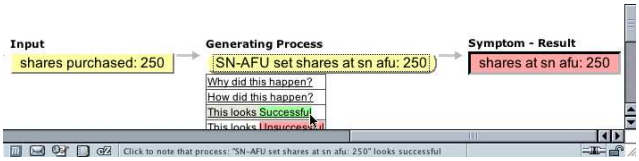

Figure 8: Noting that the stock purchase looks successful

data item, Woodstein opens a pop-up window to guide him through the rest of the process of diagnosing the problem (Figure 7). It put the objects it marked on the list of things for him to examine.

In addition to automatically identifying objects for the user to examine next, Woodstein also performs the process of elimination to help the user identify the source of the problem. If the output of a process looks wrong, but all of the inputs look right then something must have gone wrong with the process. Similarly, if the output looks wrong but the process itself and all but one if its inputs look right, then the problem must reside with the remaining input.

The next step is to determine the correctness of the process that set the shares, "SN-AFU set shares at SN-AFU". Moving the mouse over the button for the process updates the debugging trail window which provides the employee with some guidance. If the problem looks like it started before the process, the process should be marked as correct. The employee notes that the input share quantity was also 250, so he marks the process as correct (Figure 8). Woodstein, in turn, narrows the problem to the input and updates the data-history view with both new annotations (Figure 9).

The employee still doesn't know why so few shares were purchased, though, so he continues. He clicks on the triangle next to the "shares purchased" data item to open its history, scrolls the history into view, and clicks on the the process that set the data item (Figure 10).

The process that resulted in shares being purchased was "Yoyodyne set shares purchased". Clicking on a process or data item causes it to be selected and its button to appear

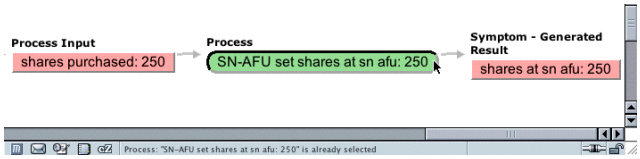

Figure 9: Viewing how the quantity of stock to purchase looks wrong

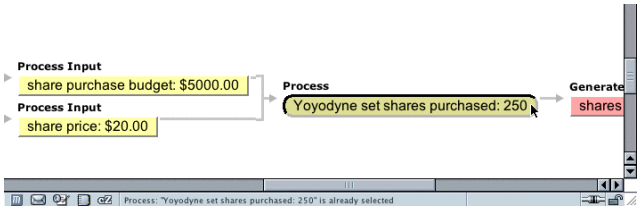

Figure 10: Viewing how the number of stock to purchase was computed

pressed in. The shares at SN-AFU were previously selected because the employee asked how they were set. Now the process for Yoyodyne setting shares purchased is selected. When an object is selected, Woodstein opens its "saved-page" view with page it saved for the object. The page is either the page the user interacted with directly, or a page the agent retrieved with the first appearance of the data item or a description of the process. In this case, the view features the saved copy of a retrieved page with the number of shares Yoyodyne intended to purchase (Figure 11). It is accessible through the "Number of Shares" label, which corresponds to the process that set the number of shares.

As in other views, Woodstein presents the data items and processes it is tracking as buttons within the saved page. Just as with other Woodstein views, the user can interact with any of these buttons to access the history of his data and the processes they were involved in.

The employee looks in the data-history view and sees the inputs to Yoyodyne setting the number of shares purchased.

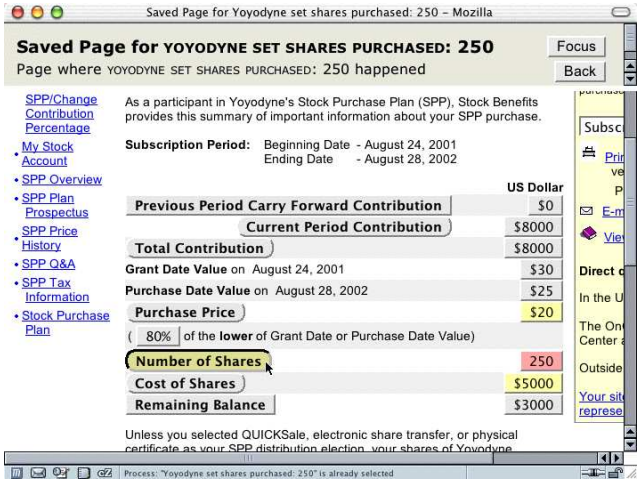

Figure 11: Viewing the saved page for the computation of the number of stock to purchase

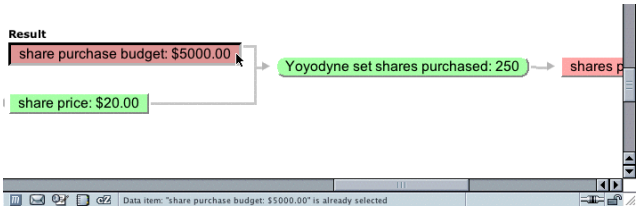

Figure 12: Viewing how the budget for the stock to purchase looks wrong

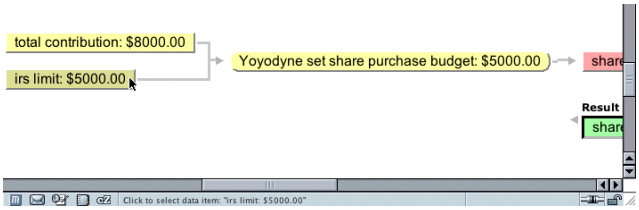

Figure 13: Viewing how the IRS limit limited the share purchase budget

Yoyodyne used the share purchase budget and the share price in computing the number of shares. Within the saved page, he can see some of the history for the share price. Yoyodyne started with two prices, the price at the beginning of the period and the price at the end of the period. Then it took 80% of the lower of the value, which resulted in the share price of $20. So far, that looks right, and the prices of the shares themselves look right to the employee so he marks them. He looks over the rest of the saved page and notices something unusual. His total contribution this year was $8000, which looks correct, but for some reason only $5000 was used to purchase shares. This is unexpected. He sees the history of the budget, and how it was computed using his total contribution in the data-history view and clicks open the share purchase budget data item (Figure 13).

Yoyodyne set the budget, and this process took the total contribution and an IRS limit as inputs. It looks like limit may be the source of the problem. The user clicks to select the limit, opening the saved page that explains it in more detail (Figure 14).

It looks like this is in fact the cause of the problem he found. This obscure and newly-introduced policy limited the amount that could be spent on his stock. With Woodstein, the employee was able to easily diagnose this problem and see that it was actually a problem with his own understanding of the stock purchase plan policies. He was able to see the history of how SN-AFU's transfer process and Yoyodyne's purchase process interacted with his own data to create the result on his SN-AFU account page. By tracing the history of the stock through the data-history view, he avoided having to look up the history of these processes on each individual site's pages. Tracing back into the history enabled him to see the exact policy that caused the unexpected result and, with the saved page view, he was able to see the explanation of the policy on Yoyodyne's site buried deep within its help pages.

The employee is happy to have identified the source of his problem. Now he wants to find out if there's an alternate program he can enroll without the restriction. He goes to the debugging-trail view and clicks on the "Complain" button

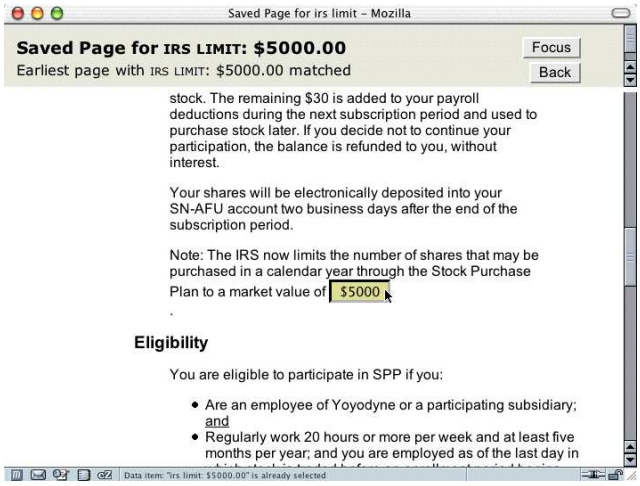

Figure 14: Viewing the saved page for the IRS limit

to generate an email (Figure 15).

Within the email's user-editable area, he asks for more information. The customer support representative (CSR) who receives this will be able to review the exact context of the request based on the path the employee took.

We can see how helpful Woodstein has been by looking at the steps the user would have to go through to find the same information without Woodstein. He would have seen the problem symptom at Yoyodyne's broker, SN-AFU. The next step would have been to visit Yoyodyne's internal pages. He would log in, find the section for his stock purchase plan account, and load the purchase history page showing how the stock was purchased, if it still was available at all. At that point he would have to go back and carefully read the help for the stock purchase plan to find the single mention of the IRS limit. Alternately, he could call Yoyodyne's internal support. He would wait on hold before talking to a CSR and, eventually, after both had traced through the process of the share purchase, he would learn about the limit.

Woodstein's records of a user's exploration and diagnosis can be useful again later, after some time has passed. Suppose that a few months later, the employee is browsing the web and loads his transaction history for his broker, Sellwell. He sees the automatic transfer from SN-AFU and

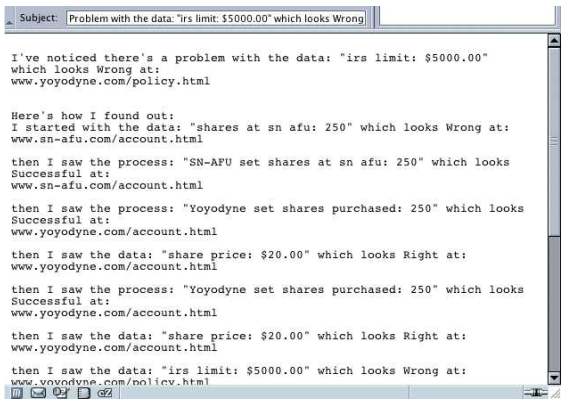

Figure 15: Complaining about the IRS limit

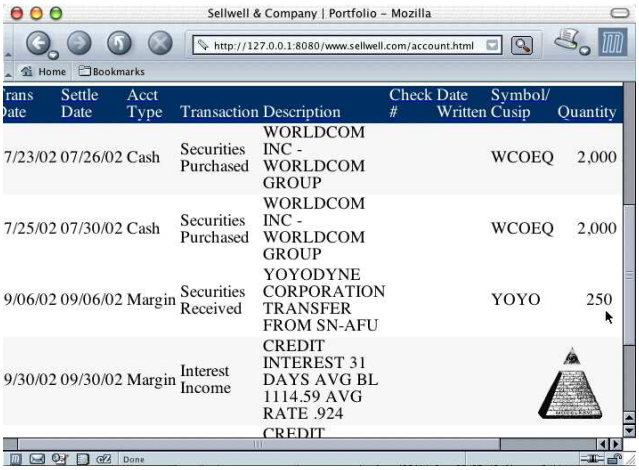

Figure 16: Viewing a stock transfer transaction

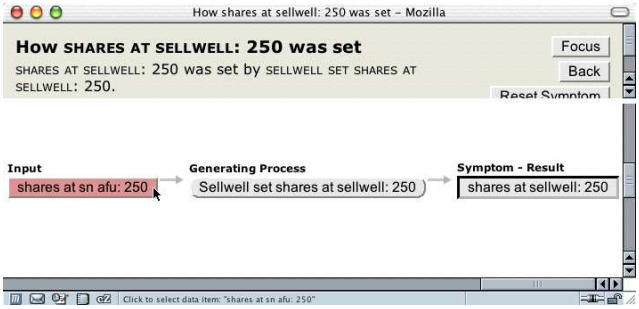

Figure 17: Viewing how stock was transferred

remembers that there was something unusual about it. He now wants to review the problem (Figure 16). The user inspects the transaction, loads the data-history view and sees that he had inspected the transferred stock when it was still at SN-AFU (Figure 17). Since he never heard back from Yoyodyne's support, he decides to contact them again.

In this example, we have seen how Woodstein presents the complete history of the purchased stock, even when it appears on multiple sites. Furthermore, we saw how it records the user's interaction with Woodstein's history record itself. The user can make use of the interaction record to, for instance, familiarize someone else with the identified source a problem. Woodstein also supports reviewing the interaction history when other pages involved in the process are visited later. Early testing has shown that even first-time users of Woodstein can perform a simple diagnosis like the one we saw in the beginning of this example. The user can access the history of some data and trace back to a particular policy involved in computing it in about 5 minutes.

4. HOW WOODSTEIN WORKS

In this section we will see how Woodstein used abstract process descriptions for the web site actions, along with annotations in the web pages, to generate the high-level interface in the example above. Support for new processes and web sites can be added to Woodstein simply by providing more descriptions like these.

Woodstein's process descriptions are formatted in a Lisp-like syntax. At the end of the scenario, the user sees the result of the stock transfer from SN-AFU to Sellwell (Figure 16). The descriptions for this transfer can be seen in Figure 18 with comments following semicolons. Woodstein models the process of a transfer from one broker to another, instantiating `<broker1>` and `<broker2>` as SN-AFU and Sellwell, respectively. A broker updating its number of shares is a process object inspectable by the user, and this update, or "set", event triggers the "meta-action" of Woodstein's updating its record of the user's shares. Note also that each process description includes a "source" page used by Woodstein to verify that the process occured.

```
(def-process-model
  <broker1>-transferred-shares-to-<broker2>
 (subject "<Broker1>")   ;; entity performing action
 (verb transferred)    ;; action performed by entity
 (object "shares to <Broker2>")
 (source <broker2>-account-page)   ;; page to verify
 (steps
  <broker2>-set-shares-at-<broker2>
  <broker1>-set-shares-at-<broker1>))

(def-event-model <broker2>-set-shares-at-<broker2>
 (subject "<Broker2>")  ;; entity performing action
 (verb set)    ;; action performed by entity on var
 (object "shares at <broker2>")       ;; var to set
 (source <broker2>-account-page)  ;; page to verify
 (step         ;; symbols used for plan recognition
  (<broker2> set shares-at-<broker2>))
 (meta-actions        ;; any model-changing actions
  ws-set--shares-at-<broker2>))

(def-meta-action ws-set--shares-at-<broker2>
 (function-sym ws-set)     ;; model-changing action
 (arg-forms   ;; arguments to model-changing action
  (shares-at-<broker2>       ;; var to set value of
   shares-at-<broker1>     ;; var to get value from
   <broker1>-account-page)))  ;; page to verify var
```

Figure 18: How the stock purchase budget was computed

Woodstein tracks web site actions to confirm that they match its process descriptions. When it expects an action to have occured, it loads the source page for the action and compares the data in the page, such as stock quantities, with the values it is represents in its own model of the process. It then visualizes the model for the user, including descriptions of processes and data and their relationships, as well as the saved pages it used in verifying and extending its model.

This example illustrated a process particular to an organization, Yoyodyne, and thus information in pages related to the process must be annotated. More common processes, such as the stock purchase and transfer, can be described in a generic way. One area of future research is to extend Woodstein to robustly extract information from unannotated web pages for common processes like these and even allow end-users to train it to track simple processes.

5. EVALUATION

We tested the Woodstein's integrated view and complaint generator with 16 subjects (see [17] for more details). We hypothesized that subjects who used one of Woodstein's views would be more successful in diagnosis and diagnose problems more rapidly. The eight subjects in the control group diag-

nosed problems spread across multiple pages of multiple web sites, then simply complained about the relevant data item or process. The eight subjects in the experimental group used one of Woodstein's views, its "process-history" view to see a history of the overall process in identifying the data item or process to complain about. This view is organized to show the history of a process featuring the data, rather than the history of data featuring processes, but is otherwise very similar to the data-history view we saw.

Participants were told how the use the system and saw it demonstrated. This took 5 minutes for control group and 20 minutes for the experimental group. They then had 10 minutes to take a "quiz" in which they used the agent to select a particular data item and complain about it. For the experimental group, this required accessing a saved page through one of Woostein's history views. Participants then had 20 minutes to solve a "test" problem involving a simulated user's data and an organization's policies. In this problem, each participant played the role of a student who is unable to graduate from his educational institution and attempts to identify the exact reason and policy involved. The experimental group could use Woodstein's view, while the control group used only pages on the web site.

All participants in the experimental group of the user study were successful in diagnosing the test problem, taking an average of 5 minutes. Only two participants in the control group were successful, requiring an average of 16 minutes. In addition, at the end of the experiment session, we asked participants in the experimental group to rate different aspects of their experiences with the agent and whether they'd use it for problems they might encounter. Interestingly, half would have used the agent to diagnose a problem if they knew it were to take longer than 5 minutes on the phone. Of the remaining four, three would use the agent if they knew a phone resolution would take 15 minutes.

Reflecting the difficulty of managing information about credit card purchases, half of the participants in the experimental group said they "strongly agree"[3] that they'd like to have an agent like this one to help in visualizing and managing their credit card transactions. Two said they "agree" they'd like to have it, and the remaining two participants discussed their concerns later in a free-from section. One noted the potential for privacy problems and the other preferred a more interactive process for complaining. Though a CSR is interactive, no CSR is able to provide all perspectives of a transaction involving multiple vendors, as Woodstein can. An area for future work is to extend end-user Woodstein to communicate with a "support" version of Woodstein, perhaps with more detailed models of an organization's processes.

In fact, one of the original motivations for Woodstein is the state of current support involving limited, text-based media including phone conversations and email. Even a perfect interaction with customer service by phone still lacks the benefits of a web interface. In order to resolve a problem, a customer must be sure to call during the hours support is offered, and have a block of uninterrupted time. The web, on the other hand, is always available and if some information is not immediately at hand, a decision or investigation can be postponed. Furthermore, it is often easier to understand complex information when it is presented graphically[10]. All of the advantages of a slick web site's presentation are lost when the customer has to hear his options or receive instructions over the phone. In fact, all of the advantages of a digital format are lost. When both the user and a CSR use an agent like Woodstein, each could highlight and talk about data items and processes, and even hypothetical possibiliites and future events.

[3]Participants expressed their level of agreement with a 7-point Likert scale: 1=strongly disagree, 2=disagree, 3=somewhat disagree, 4=neutral, 5=somewhat agree, 6=agree, 7=strongly disagree

6. MANAGING HYPOTHESES WHEN DIAGNOSING PROBLEMS IN SYSTEMS

We see the possibility of user annotation as more broadly applicable beyond just e-commerce on the web. Computer users interact with systems everyday and often run into problems. When a problem is repeatable or particularly troublesome, a user may send a bug report in an informal way. Systems like Bugzilla[11] have been developed for automatically managing users' bug reports. Additionally, some applications, like Mozilla[12], "close the loop" and automatically send a bug report when erroneous behavior is detected, such as when the program crashes. These applications often allow users to provide some free-form information about what they were doing when the problem occured, but they don't support user diagnosis in general. Unlike a developer, however, a user is in the perfect position to diagnose hard-to-find bugs involving complex configurations. Further, we suspect that many users would prefer to take a few moments to diagnose a configuration problem with their operating system then have to reinstall the entire system and all of their applications. In fact, often users have to do exactly that and reassemble the history of what they installed to identify a conflict. Of course, perfectly developed software with no bugs would be ideal, but other factors govern the adoption of new software. Regardless, a higher-level interface for managing hypotheses during diagnosis would be helpful.

7. RELATED WORK

No other general system explicitly supports end users in visualizing and diagnosing problems with their e-commerce transactions or other actions on the web. An earlier work [9] explains the problem domain in more detail, discusses the broader issue of end-user debugging and describes research in software debugging that has influenced this work.

7.1 Debugging

ZStep[8] is a debugger designed to support the cognitive processes in debugging software, especially visualization and localization. ZStep is reversible—it records every step of the execution of a program and allows the programmer to replay it backward as well as forward.

Woodstein builds on these features and extends them to the domain of web processes. Like ZStep, Woodstein records a complete history of a process and allows the user to jump back to any point of that history or replay it forward or backward. While ZStep tracks the graphical output of the program, Woodstein tracks the pages generated by the process. The user can go from a data item in a page to the point in the process in which it was set, and vice versa.

Some researchers have focused on how to better support end-users and novice programmers in debugging. In the area

of "end-user software engineering", the Forms/3 project in particular supports end-users in debugging spreadsheets[15].

7.2 Program Slices

Woodstein performs program slicing to generate its audit trails. Audit trails are used in business to track the history of a record and show all of the processing it has undergone. Similarly, program slicing is a software engineering technique for focusing only on the steps of a program that affect the value of a particular variable[18]. It is helpful for debugging, when a programmer knows a variable has the wrong value and wants to know how it was computed.

Program slice tools typically highlight the lines of code, modules or files in a slice[1]. This is useful for programmers, for whom the source code is the primary representation of the program. Within the domain of web actions, however, we don't expect the abstract models to be particularly meaningful to end-users. Rather than presenting the abstract description of the process, Woodstein generates explanations of the process' actual concrete execution.

Some tools present slices via control-flow graphs or program dependency graphs[6]. Woodstein presents the program dependencies in the data-history view, and the program execution tree in the process-history view.

7.3 Capturing User Annotations

Little research has focused on capturing user annotations. Most systems, such as Margin Notes[13] record free-form annotations with no representation of their meaning. Woodstein features *effective* annotations with meaning, enabling it to manage the annotations and even create new ones.

Trellis is a system that supports user annotation of objects in an application's interface[5]. Annotations in Trellis are typically used to indicate the original context of some information, allowing users to refer back to this context.

Third Voice is a system for managing user annotations of web pages, allowing users to see pages on the web overlayed with the comments of other users[16].

7.4 Agent-based User Interface

The closest work to Woodstein's interface in style and spirit is Collagen[14]. Collagen works to guide users through tasks, such as making flight reservations or setting up a timed recording on a VCR. It tracks the steps of the user's activity, and matches them through plan recognition to discourse models. Unlike Woodstein, Collagen's typically represents the agent as a character in the interface.

Collagen guides the user through the normal operation of a system. Woodstein, on the other hand, helps the user understand what has happened when something goes wrong. Collagen supports the user's task in the present and future while Woodstein explains the past history of the user's actions.

8. CONCLUSION

We have presented Woodstein, a web interface agent for enabling end-users to visualize and understand their actions on the web. Further, Woodstein helps users manage their judgements of the correctness of their data and processes and diagnose the sources of problems they run into. We saw an example of how this record of a user's judgements can be useful both to share with others, including customer service, as well as for future reference.

9. ACKNOWLEDGEMENTS

Thanks to Marc Millier for his help in developing the example scenario, Chris Laux for help in refining Woodstein's interface and Mary Jane for inspiration.

10. REFERENCES

[1] Thomas Ball and Stephen G. Eick. Visualizing program slices. In *IEEE/CS Symposium on Visual Languages*, pages 288–295, 1994.

[2] Carl Bernstein and Bob Woodward. *All the President's Men*. Simon and Schuster, 1974.

[3] David Daniels, Corina Matiesanu, and David Schatsky. *Jupiter Consumer Survey Report: The State of Customer Service 2003*. Jupiter Research, 2003.

[4] David Daniels and David Schatsky. *Quantifying the Cost of Poor Service: Investing in Customer Service to Defend Revenues*. Jupiter Research, April 2002.

[5] Yolanda Gil and Varun Ratnakar. Trellis: An interactive tool for capturing information analysis and decision making. In *Lecture Notes in Computer Science 2473*. Springer-Verlag, 2002.

[6] Tommy Hoffner, Mariam Kamkar, and Peter Fritzson. Evaluation of program slicing tools. In *Automated and Algorithmic Debugging*, pages 51–69, 1995.

[7] Esteban Kolsky. *The Six Steps for Web Self-Service in Customer Service*. Gartner, Inc., March 2002.

[8] Henry Lieberman and Christopher Fry. Zstep 95: A reversible, animated source code stepper. In John Stasko, John Domingue, Mark Brown, and Blaine Price, editors, *Software Visualization: Programming as a Multimedia Experience*. MIT Press, Cambridge, MA, 1997.

[9] Henry Lieberman and Earl J. Wagner. End-user debugging for e-commerce. In *Proceedings of IUI'03*, 2003.

[10] Richard Mayer. *Multimedia Learning*. Cambridge University Press, 2001.

[11] The Mozilla Organization. Bugzilla: The mozilla bug database. bugzilla.mozilla.org

[12] The Mozilla Organization. Mozilla. www.mozilla.org

[13] Bradley J. Rhodes. Margin notes: building a contextually aware associative memory. In *Proceedings of the 5th international conference on Intelligent user interfaces*, pages 219–224. ACM Press, 2000.

[14] Charles Rich, Candice Sidner, , and Neal Lesh. Collagen: Applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22(4):15–25, 2001.

[15] J. Ruthruff, E. Creswick, M. Burnett, C. Cook, S. Prabhakararao, M. Fisher II, and M. Main. End-user software visualizations for fault localization. In *ACM Symposium on Software Visualization*, 2003.

[16] Third Voice. www.thirdvoice.com

[17] Earl J. Wagner. Woodstein: A web interface agent for debugging e-commerce. Master's thesis, MIT Media Laboratory, 2003.

[18] Mark Weiser. Program slicing. In *Proceedings of the Fifth International Conference on Software Engineering*, pages 439–449, New York, 1981. IEEE.