AUTOMATING THE INTERNET: Agents as User Surrogates

Bruce Krulwich,
Agentsoft Ltd.

The explosion of the World Wide Web as a global information network brings with it a number of related challenges for automation. First, nontechnical users should be able to benefit from the information available on the Web without being overwhelmed by technical detail. Second, users should be freed from mundane and repetitive browsing tasks. Third, and most critical, information from the Web should be available in the format and combination that best fit the user's task, regardless of the pages on which the information was originally found.

This report looks at these issues for Internet automation in the context of new software agent technologies that act as user surrogates for carrying out routine Web activity. Such surrogates enable automation of all interactions with HTML pages and forms—not merely the retrieval of specific URLs—and also the flexible integration of Web information into customized reports and other applications.

CHALLENGES OF WEB AUTOMATION

The tasks performed on the Internet and on intranets are as diverse as the information they contain and the people that make use of that information. Because of this diversity, and because of the inherent complexity of the information and services available, the goal of automating routine Internet behavior is formidable. Consider the following routine Internet tasks:

- scanning a dozen newspaper sites each day to gather articles relating to a company's business into a single report for executives;
- looking up price and recent performance on stock leads, collecting profiles on companies from evaluation services, and reading recent relevant articles from online newspapers and magazines;

- adding permission information to an intranet server, printer drivers, an Internet firewall, and the local file servers for a new user to the office network;
- scanning online newspapers and magazines for articles that mention a company's clients or competitors, using the latest client and competitor databases on the company's intranet.

Each of these examples raises a different challenge for automation. The first, scanning online news for a fixed set of keywords, is by far the simplest, and in fact is a service offered by a number of Internet service companies. It is, however, a task that cannot be handled directly by "scheduled download" software because the user must fill in forms and collect results rather than have fixed URLs regularly downloaded. Additionally, the recipient executives may want a news report in a particular format, based on their business, rather than a collection of articles.

The second example, collecting a variety of information about a given stock, illustrates the need to automate tasks that are performed on demand with minor variations (in this case, the name of the company), rather than on a fixed schedule with no variation. It also requires that highly disparate information, such as data, graphs, and articles, be gathered into a single common report.

The third example is the most unusual one in that it doesn't involve collecting information, but rather using Web-based forms to carry out specific actions on a network. This again highlights the need to perform arbitrary online functions rather than simply gather Web pages.

The fourth example, searching online news for companies mentioned in a company's client and competitor databases, involves using information from a private source, perhaps requiring customized programming, to drive information search on the Web.

From these four examples we can see the following desiderata for Web automation:

- automate arbitrary Web activity, including filling out forms;
- gather information from various Web pages into a single report;
- act on a scheduled basis or on demand;
- integrate information from a variety of sources, both private and public; and
- smoothly operate with custom program modules.

These functions clearly go beyond the simple automation technologies found in today's download systems or in products for information push. At AgentSoft, these Web automation goals motivated the development and use of AgentSoft's LiveAgent product.

USER SURROGATES

The key feature of LiveAgent's agents (hereafter "LiveAgents") is that they can act on the Web as surrogates for the user. Virtually any activity that a user can perform in a Web browser can be automated by a LiveAgent. Furthermore, LiveAgents carry out the activity in a fashion that exactly simulates the user so that servers on the Web cannot distinguish the agent from the user. Agents can have parameters, such as names or keywords, that are used in the course of this activity as inputs to forms or to otherwise control the agent's activity. These agents can act either on demand or according to a schedule.

In the first example above, an agent could be scheduled to execute every morning, say at 6:00 a.m. The agent would go to the search forms for a dozen online news sites, enter a fixed set of keywords, and collect the articles. The report would be waiting each morning for the user's arrival, and could also be sent automatically to the recipient executives or made available on the company's intranet.

In the second example, the agent could use a company name as a parameter to retrieve the desired information from each of the online information providers. The agent would then insert the data, graphs, and articles into a report template designed for that purpose. The agent could automatically adapt to unavailable Web sites by trying alternative information sources.

In the third example, a user could create an agent that takes the name and status of a new employee, goes to the intranet-based device management page for the various network subsystems, and enters or edits the employee's data at each site. Through the agent, the user would have a single point of entry to all of the systems without having to reenter the information for each one.

In the fourth example, a user could have an agent that uses customized Java methods to retrieve the company's clients and competitors from a database, and proceeds to search Web-based news sources for each of them.

In each of these cases agents act as user surrogates in accessing the Web, a corporate intranet, or both. These agents access pages or forms, gather information into customized reports, and utilize arbitrary Java code as needed.

AGENT CREATION IN LIVEAGENT

Using LiveAgent, a developer can easily create Java-based agents that gather information from the Web. While the developer browses, LiveAgent's recording mechanism "learns" the actions taken: which links were clicked, what was typed in a form, and so on. The recording mechanism acts as a kind of high-level macro recorder for creating Web automation scripts. The developer can also specify that some items will be changeable each time the agent is run based on the values for the agent's input parameters, and can assign these items new values before each run for the agent to use when it browses. Changeable items include the contents of text boxes, radio boxes, and list

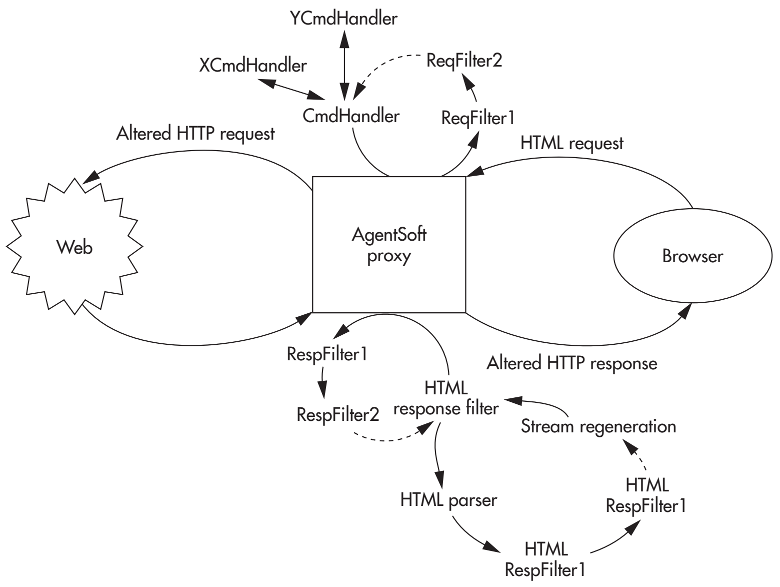
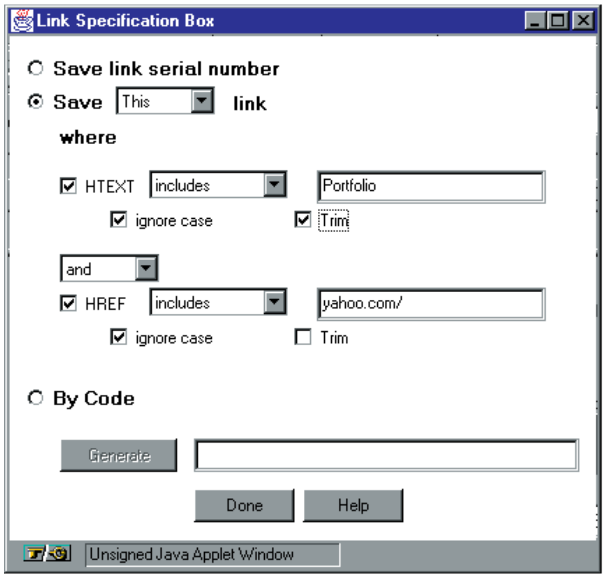


Figure 1. The architecture for the AgentSoft proxy server.



- ○ Save link serial number
- ⦿ Save [This] link

where

- ☑ HTEXT [includes] [Portfolio]
  - ☑ ignore case ☑ Trim
- [and]
- ☑ HREF [includes] [yahoo.com/]
  - ☑ ignore case ☐ Trim
- ○ By Code

Figure 2. Link specification box for selecting use.

boxes. Lastly, the developer can specify which pages or portions of pages to include in a report.

While a developer records an agent, LiveAgent's agent engine, also called the AgentSoft proxy, alters the Web pages being browsed so that user events can be monitored and recorded. The proxy monitors browsing sessions and inserts appropriate code onto browsed Web pages. This involves adding and routing event handlers, such as "onClick()" events for links, buttons, and input fields to the appropriate recording function. The activity occurs in the main browsing frame. Hidden frames store data and functions that persist across pages.

The AgentSoft proxy effectively sits between the browser and the Web (see Figure 1). Unlike conventional proxies, which are generally used to implement caching or to provide more secure network access, the AgentSoft proxy is designed to allow arbitrary filtering of HTTP requests and responses via dynamically loaded Java classes. A design theme repeated throughout the proxy is that of dynamic filter chaining—a set of easily configured classes, loaded at runtime, that sequentially handle different filtering functions.

When an HTTP request header is passed from the browser to the proxy, the proxy forwards the request to a chain of request filters, which alter the request arbitrarily. Request filters can carry out tasks as subtle as changing one or more of the HTTP request fields. They can also completely redirect the requested URL to a new URL or, more drastically, create the resulting HTML (akin to a client-side CGI script) without querying the Web at all.

When data returns from the Web in response to a query, the proxy passes the response header and the byte stream to the response filters. Like their request counterparts, response filters optionally modify the information according to their programming. Response filters can either modify the response header (that is, change the response code or other header fields) or filter the actual body of the response. Coupled with this is an HTML parser that enables response filters to process at the HTML level rather than at the character level. After all of the appropriate response filters have processed the stream, it is passed to the browser as the result of the original URL request.

The use of the proxy in this capacity enables LiveAgent recording to be as close as possible to normal browsing for the user, and indifferentiable from normal browsing to the Web site whose pages are being browsed.

HTML POSITION DEFINITION LANGUAGE

Whenever the user records clicking on a link, the proxy must try to understand the user's action for future replay. The same user action, for example, could be understood as clicking on the fifth link on the page, or the link with the word "profile" in the link anchor text, or the second link after the first image, or any number of other generalities. LiveAgent's approach is to ask users to specify their intentions using AgentSoft's HTML Position Definition Language (HPD), as shown in Figure 2. HPD is a simple, flexible expression syntax that allows the specification of an arbitrary HTML element on a page.

The HPD language is also used to specify regions of the browsed pages that should be included in the agent's report. Report items are extracted from the browsed page based on two HPD expressions, one for the start of the region and one for the end. For example, a developer recording an agent to retrieve stock information could include a company's stock price in a report by specifying the region starting with "the <nobr> tag in a table cell whose text starts with 'Last price'" and ending with "the first </nobr> tag after the start." An agent's set of result items are collected into a single report for the agent user either as a simple list, by insertion into an HTML template, or through execution of developer-specified Java code.

The HPD language is also used to specify conditional branching and looping logic for an agent. Agents regard each set of actions that take the user away from the current page as a task. Each task can have a conditional associated with it to determine whether or not the task should be performed.

Consider Web browsing as a tree, with each Web page a node on the tree. Each node could contain subtasks, such as other links to follow or forms to complete. These subtasks can have associated conditions for executing the task. For example, "If it is Sunday, follow the link to the crossword puzzle and retrieve it." If it is not Sunday, then the link is simply not followed and another node is visited. There can also be loops with tasks ("Download stories while there are still stories about the Internet") and interdependencies with conditions ("If the Sunday puzzle was downloaded, then download the sports section, too").These conditions are specified using HPD expressions.

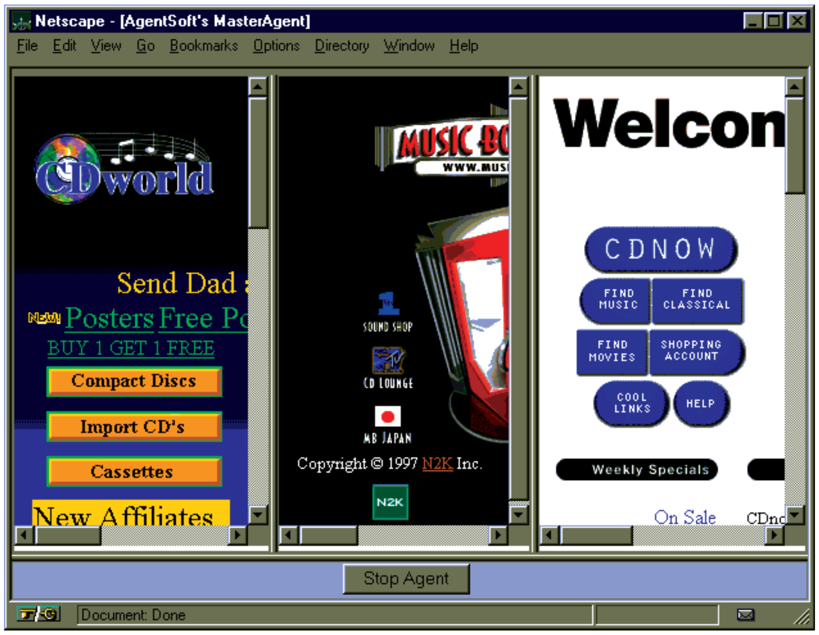


Figure 3. MasterAgent running three agents in parallel.

In short, the HPD language provides a mechanism for user specification of an agent's conditions, actions, and results. The language can also be used programmatically by Java developers extending the capabilities of a LiveAgent. In this way the language enables agent developers at any stage to refer to HTML content in a standard, clean, and efficient manner.

THE MASTERAGENT TOOL

MasterAgent is a developer's tool that uses a set of mini-agents (actual LiveAgents) to collect information in parallel (Figure 3) , and then merges the information into a single table for reporting to the user. MasterAgent provides a framework for running multiple, parallel agents from a single point of entry. Additionally, it allows the developer to insert value-added Java classes for more intelligent filtering of results.

This tool is particularly useful in applications that require collating similar information from a variety of sources. For example, a client planning a business trip wants to compare data from five different airlines the client frequently uses for a particular route. Specifically, the client wants to compare prices, stop-over information, and departure times from all five airlines. This scenario requires an agent with more intelligence, such as being able to compare alternatives rather than simply report them. It must be able to handle the specific question "Which choice is best for me?"

To define a MasterAgent, the developer provides an input form, a set of mini-agents, a description of an output table (including the set of fields and their types), and various control fields. MasterAgent prompts the user for the variables needed by all of the mini-agents (such as departure and arrival destinations), and then launches the mini-agents in parallel.

For each mini-agent, the developer provides a class that maps input from the MasterAgent to the specific mini-agent, and an output mapper mechanism that takes output data from each mini-agent and projects it to the rows of the output report table. These input/output wrappers adapt the standard LiveAgents for use in the MasterAgent.

Welcome to the Software Solutions Group, a division of

A world of enabling technologies is now available at a computer near you:

http://www.bell-labs.com/org/ssg

Or call the Software Solutions Group at 1-800-462-8146 or 415-943-4076 and ask for Dept. T12W to receive a free product catalog.

As the agents report their data, MasterAgent merges the similar, tabular-oriented data into a single report table. This resulting merged report is far more friendly and usable than the five individual reports from the five individual mini-agents. More importantly, the developer can write table-processing filters in Java to arrange the output table into the desired format for the user. Through these table filters developers can add custom intelligence to their agents.

By default, the MasterAgent tool provides routine sorting/merging capabilities. The MasterAgent developer needs only to insert the value-added sorting or marking criteria to create a more complete, intelligent agent.

LOOKING TO THE FUTURE

To date, AgentSoft has emphasized four target functions in the development of LiveAgent:

- the easy record and replay of agents,
- the definition and integration of the HPD language,
- the ability of agents to include branches and loops, and
- the hooks for easy and focused development of Java code that integrates seamlessly into LiveAgents.

As the Web grows, the amount of available information valuable to a wide variety of users increases. Achieving the most benefit from this information requires making it easily accessible, on demand, to users of varying technical backgrounds, and integrating it into user-centric reports. AgentSoft's LiveAgent is a development toolkit that makes this possible, and we expect it to grow in power as the Web continues to develop.

The primary ongoing research involves better integrating LiveAgents with other applications and developing agents that can reason about the information a user seeks and find Web sites where that information is available. ■

ACKNOWLEDGMENTS

Thanks to Jeff Rosenschein and Joe Weisblatt for their perspectives on LiveAgent technology and their assistance in writing this article, to Deborah Weisblatt for contributions to the article, and to the whole LiveAgent team for making the product a reality.

Bruce Krulwich is a senior research scientist at AgentSoft Ltd., where he directs the Advanced Technology Group. Prior to joining AgentSoft he was a research scientist at Andersen Consulting's Center for Strategic Technology Research, where he was the principal investigator for the intelligent agents project. He received his PhD in artificial intelligence from the Institute for the Learning Sciences at Northwestern University.

Readers can contact Krulwich at AgentSoft Ltd., 38 Pierre Koenig Street, Jerusalem, Israel, or by e-mail at brucek@agentsoft.com. More information about AgentSoft is available at http://www.agentsoft.com.