A Visual Approach to Internet Applications Development

Mauro Mosconi – Marco Porta
Dipartimento di Informatica e Sistemistica – Università di Pavia
Via Ferrata, 1 – 27100 – Pavia – Italy
mauro@vision.unipv.it – porta@vision.unipv.it

1 Do Advanced Users Wish to Build and Control Their Own Internet Applications ?

To allow Internet users to find, collect and manipulate information available on the Web, different solutions have been developed (IEEE 1997) by researchers and software companies which aim at simplifying the interface as much as possible (search engines) or even acting on the users' behalf (software agents). Nevertheless, there is a range of applications where the overload involved in training an adaptive intelligent system would be unacceptable, while a traditional browsing approach would result in tiresome, time-consuming effort.

To tackle this class of applications, we propose a data-flow visual environment in which non-naive users can accomplish their goals better and more rapidly by directly composing simple visual programs themselves, while preserving a sense of control over the system. In the following, a simple example application is discussed to highlight the potential of our approach.

2 The Development Platform

The system we use for building our visual Internet applications is VIPERS (Ghittori, Mosconi and Porta 1998), a general purpose, visual programming environment based on an augmented data-flow model (Hils 1992) and developed at the University of Pavia. VIPERS uses a single interpretive language (Tcl) to define the elementary functional blocks (the nodes of the data-flow graph). Each block corresponds to a Tcl command (or procedure): such a command may itself invoke the execution of other programs as subprocesses.

Published in the Proceedings of the 8th International Conference on Human-Computer Interaction (HCI'99), Vol. 1, 22-27 August 1999, Munich, Germany.

VIPERS elementary modules have a square shape and present connection points, or ports, on their lateral sides; programs are assembled through direct manipulation, by positioning and properly connecting the available modules: entire programs can therefore be built without typing any line of code.

3 The Visual Approach

Figure 1 shows an example of a visual program built through VIPERS. Its purpose is to explore the Web to find a set of E-mail addresses (to which a message will be sent later on), starting from one or more words and/or sentences. To avoid sending messages to persons out of target, the program also generates a form, to be filled in by the user. For each address, this form shows its context (the ten words preceding it and the ten words following it) and a check box, used to select or exclude the address. The form is subsequently processed by a CGI program. In the example, you may note both ad-hoc blocks (purposely prepared for this kind of application) and more general blocks, which can be used in other application domains as well, for building general-purpose programs.

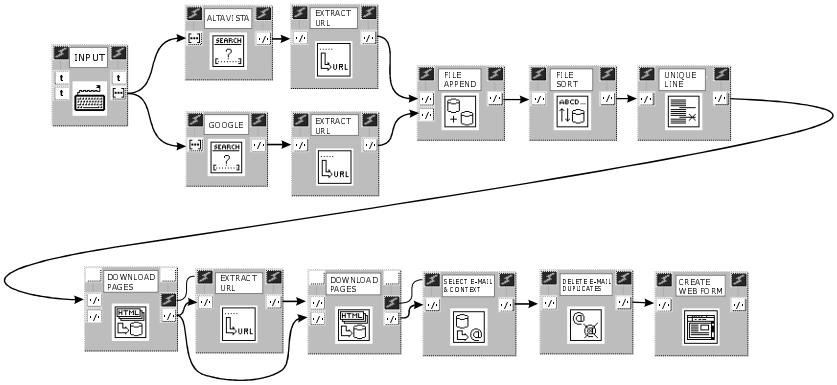

Figure 1: an example of visual program built through VIPERS

Each input/output port in VIPERS is characterized by a special icon indicating the corresponding data type. Data types used in our example are: t (text), [...] (list) and •/• (file path). It is to be noted that (at least in this example) when a module gives out a file path name, this path refers always to a temporary file.

The first module in the figure (block INPUT) allows the user to enter the words/sentences to be searched for. These words/sentences are then provided, in parallel, as inputs, to blocks ALTAVISTA and GOOGLE, which connect to their relative search engines and produce HTML pages (local temporary files)

containing the search results. Blocks EXTRACT URL analyze these results, locate the links they contain and create text files with one link per line. The purpose of block FILE APPEND is to link together the files it receives as inputs and to produce a single file. In the example, therefore, it generates a file holding all the links present in both the Altavista and the Google search results. This file is then alphabetically sorted by block FILE SORT and "cleaned up" by block UNIQUE LINE, which eliminates any duplicated lines (links).

At this point, a quite meaningful set of results should have been obtained (which could be exploited to automatically download the sites' pages, through already available software tools). In our program, now the information access phase starts. For each link in the temporary file at the output of block UNIQUE LINE, the following operations are performed:

- the page corresponding to the link is accessed (downloaded);
- the page is appended to a buffer;
- the page's links are in turn extracted and the relative pages are then accessed and appended to the buffer (*two-level spidering*).

These operations are accomplished by blocks DOWNLOAD PAGES and EXTRACT URL. The first DOWNLOAD PAGES (macro) block in the figure receives, as input, the file of the links whose pages are to be downloaded and yields a file (buffer) containing all these pages as output. Actually, this block has another input port as well: if a file path name is provided for it, such a file is assumed as an initial buffer to which the pages must be appended. We assume that the default input value for this port is an empty file: VIPERS, in fact, allows default values for the inputs of a block to be easily specified, which are used when no data is explicitly provided.

In the figure, you can also note a thin line without arrows connecting the *control* ports (those with the lightning symbol) of blocks DOWNLOAD PAGES and EXTRACT URL. This line is a *control signal* and is activated only when the download of all the pages has been completed. Control signals are used in VIPERS to achieve correct synchronization: if a signal exists between an output control port (on the right) of block A and the input control port (on the left) of block B, then the execution of block B can not occur before the whole execution of block A.

Returning to the example, when all the first-level pages have been downloaded, the links they contain are extracted by block EXTRACT URL and provided for the second DOWNLOAD PAGES block. This block also receives the output of the first DOWNLOAD PAGES block as an initial buffer and produces a single file containing both the first-level and second-level downloaded pages. The specialized module SELECT E-MAIL & CONTEXT receives this buffer as

input and singles out the E-mail addresses contained in it. For each address, it then yields, besides the address itself, the title of the page in which it was found and its context as output (the ten words preceding it and the ten words following it). Lastly, block DELETE DUPLICATES eliminates any duplicated E-mail addresses and the block CREATE WEB FORM generates the web form to be checked by the user.

As already stated, block DOWNLOAD PAGES is a *macro* block, that is, it is composed in turn of other blocks. Figure 2 shows its internal structure.

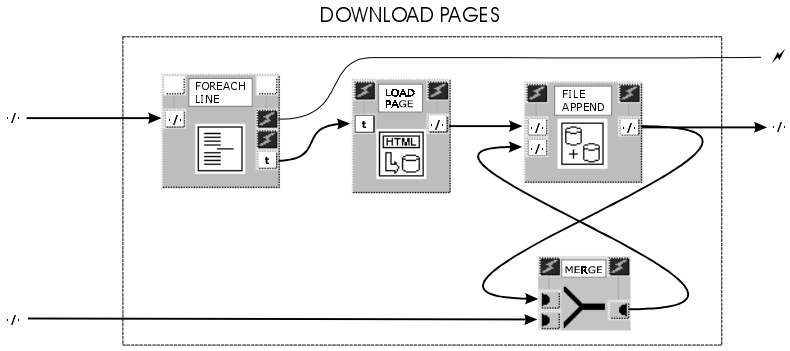

Figure 2: the internal structure of block DOWNLOAD PAGES

As one can see from the figure, a file path name is provided as input to block FOREACH LINE. This block, in turn, is a macro block implementing the *foreach* control construct, allowing elements of a sequential data structure (in this case, a file) to be analyzed and made available, one after another. Block FOREACH LINE emits the various lines (links) of its input file, as they are taken out, but only at the end of this process does it activate its output control signal. Block LOAD PAGE connects to the URL it receives as input and downloads the corresponding HTML page on a temporary file, whose path name is made available as its output. The MERGE block fires when either of its two input ports receives a new data item, which is then emitted as an output. Here, it allows an initial buffer path name to be specified, to which, during the subsequent iterations, the various HTML page contents will be added by block FILE APPEND.

The whole output of the program of figure 1 is a Web page (a form) like that shown in figure 3, which could be delivered via E-mail to the user. By displaying the form in his/her browser, the user will then be able to select the addresses, excluding, for example, those of the various webmasters, etc.

Figure 3: an example of form generated by the program of figure 1

By pressing the SUBMIT button, the form data is passed to a CGI program on a server, which processes it.

4 Conclusion

Our experiments indicate that a general purpose data-flow visual programming environment like VIPERS can be effectively used by skilled users to digest Web information in an easy manner. The program described above was used as a testing exercise with a group of six engineering students. All the testers were able to assemble the application from the modules (blocks) and macromodules shown in the figure.

It is worth noting how the visual programming approach described here could be used for rapid program prototyping as well. Indeed, once a program has been built, it could be advantageously translated into a single script (for example in Tcl), which can be used without need for the VIPERS visual interface.

We are now developing an on-line version of the VIPERS system, which will allow to build visual Internet applications by simply connecting to a proper site and assembling ready-made blocks taken from various libraries. Block execution will occur either locally or, when needed, on remote machines.

References

Ghittori, E., Mosconi, M., Porta, M. (1998). Designing new Programming Constructs in a Data Flow VL. *Proceedings of the 14th IEEE International Conference on Visual Languages* (VL'98, 1-4 September 1998, Nova Scotia, Canada).

Hils, D. D. (1992). Visual Languages and Computing Survey: Data Flow Visual Programming Languages. *Journal of Visual Languages and Computing*, 3, 69-101.

IEEE Internet Computing, vol. 1, n. 4, Jul-Aug. 1997.