Implementing Electronic Forms with the World Wide Web

by George P. Pipkin

A strategy at the University of Virginia to use electronic forms to reduce the size of its data entry operation has led to the development of a prototype of an electronic forms system based on World Wide Web technology. This article describes the elements of the system, its structure, and its implementation to date.

Along with many other large institutions, the University of Virginia (U.Va.) is actively looking for ways to simplify its business processes and reduce the overhead associated with administration. One of the most important tools that we can use to work toward this goal is the University’s high-speed data network. A fundamental strategy we hope to employ is to convert much of the paper that people work with on a day-to-day basis to digital formats on the network, and in doing this nurture automation and process simplification while increasing the overall speed of information flow throughout the organization.

The University’s administrative sector still relies on paper-based objects of work that are moved around the institution through a manual messenger-mail system. The majority of these objects are forms and reports. More than 1,200 different types of printed forms are used institution-wide, and every month our administrative mainframe produces more than 700,000 sheets of paper reports.

In 1994, the University embarked on a plan to reduce the resources needed to run its administrative mainframe. A key strategy associated with this effort was to use electronic forms to reduce the size of its data entry operation. The University’s Information Technology and Communication (ITC) department examined several commercial electronic forms packages in great detail, but found that they fell short of our needs in a number of ways. Some of them were only available for Windows platforms. Most of them relied on e-mail as the transmission medium, compromising the system’s overall robustness and reliability. All of them required a high-maintenance environment to be fully functional, both at the workstation and at the server level. Because we assessed the potential range of use as a key factor in the ultimate impact of this strategy, introducing a solution that required substantial maintenance was unacceptable.

Since the emergence on several platforms in the fall of 1994 of World Wide Web (WWW) browsers\(^1\) that fully support electronic forms features, many users of the Web have become accustomed to encountering data-input fields on the pages they see. However, fewer people realize how this capability could be put to use to construct a full-fledged enterprise electronic forms and work flow system.

The capability Web browsers deliver is based on openly defined protocols. Browsers have been written that support these protocols on a wide range of platforms, and new browsers are appearing every month. Increasingly, a Web browser is becoming a base component of the personal computer operating system.

George P. Pipkin (gpp8p@virginia.edu) is a technical advisor in the Office of Information Technologies at the University of Virginia (U.Va.). He has been involved with personal computing since 1975, when he built his first computer from plans published in Radio-Electronics Magazine. Pipkin played a major role in developing U.Va.'s strategy to use Web tools in support of electronic forms and other reengineering efforts. He is currently working in the Advanced Technology Group on a project to use the Web for document management and imaging. His home page is http://holmes.acc.virginia.edu/~gpp8p/home.html

\(^{1}\) A browser is a program that allows a user to read hypertext by giving some means of viewing the contents of network nodes and of navigating from one node to another. Examples of browsers for the World Wide Web include Mosaic, Lynx, and Netscape; they act as “clients” to remote “servers.”
electronic forms and work-flow packages can't match.

Web browsers are extremely flexible: they can display a wide variety of objects directly, and because they are MIME-based, these clients can coordinate the use of specialized viewers to handle objects with which they are not directly familiar. When this flexibility is combined with their native electronic-forms capabilities, the result is a versatile work-flow client.

Web "pages" that you see in your browser actually are stored on a WWW server. In the case of interactive Web pages, there is a lot more happening on the server end than simply transmitting an HTML document that is stored there. Special "handler" programs are necessary to respond to whatever the user puts into the page. While a variety of generic handlers have emerged that will e-mail information that comes from Web-based forms to a specified address, they fall short of what is required to support work flow. This includes management of an extended dialog with the user, control of access to different functions, and the ability to "route" objects from one user to the next in an organized way.

Last year, the University of Virginia set out to implement an infrastructure that would allow us to surmount these challenges and use WWW technology in support of its process simplification goals. WWW clients and servers are being used initially to create an enterprise electronic forms "front end" (user interface). We also hope to use the same combination to deliver reports electronically over the institution's broadband network. The anticipated outcome is a major contribution to an institutional program aimed at streamlining its administrative processes and ultimately redirecting resources to more directly support instruction and research.

Infrastructure elements

For Web browsers to be used to meet the needs of colleges and universities for full-fledged electronic forms and work flow, a number of missing components will have to be developed.

Handler program generation. The most obvious need is for a facility that will digest the HTML code associated with a form and generate the "back-end" handler components specific to that form. Servers must be capable of delivering blank forms to the user Web client, transferring data from filled-in forms to a centralized DBMS record, and taking the data out of that record and putting them back into a form so that they can be edited or reviewed.

Point-and-click designer tool. Users need a user-friendly tool that will allow them to create new forms without having a prior knowledge of HTML, or other specialized programming languages. Even though workstation-based tools are available, they are specific to particular platforms, and require uploads to move the resulting forms to the server. The tool described in this article is server-based and will integrate seamlessly with other components of the forms infrastructure.

The in-box. A mechanism is needed to present a virtual in-box that will provide easy access for individuals who need to review or approve forms.

Authentication. Individuals filling out or reviewing forms must be able to identify and authenticate themselves to the system and not have to supply a password with every page.

Business rules. There must be a mechanism to check the information coming out of electronic forms and ensure that the entries comply with organizational rules and policies.

Routing. There must be a routing engine that will move the forms to the places where they require review, retaining a log of everything that happens, and providing an easy way of checking on a form's current status. The routing engine must be capable of delivering forms to the points of entry for enterprise transactional systems once they have passed through the business rule filters and the approval chain.

Transactional system point of entry. There must be an easy way to format data for transfer to transactional systems and a mechanism to effect these transfers in an organized and scheduled way.

The system's structure

The University of Virginia's electronic forms system uses special handler programs to manage the interaction between the user and a central database that resides on the WWW server. Each handler program corresponds to a particular form. The programs are actually perl scripts, and they are produced by a code generator that processes the hypertext markup associated with the form and creates the necessary specialized handler program that runs on the Web server. This program accepts data coming in from WWW forms being submitted from client machines and conveys these data to tables in the database that are specific to particular form types or other objects. When the user wishes to edit a particular form, the handler retrieves these data and places them into a form that is sent to the appropriate WWW workstation client. This handler component also implements the form's associated business rules and carries out routing functions.

Access to individual forms is facilitated by a "virtual in-box" that is stored in the central data-
Only those forms or other objects that are currently associated with a particular user are seen in his or her in-box. Forms are visible in one in-box and then another as they are routed through their approval chain. When a form has reached the appropriate stage in its routing process, its handler transfers information to transactional systems through a kind of “data pipeline.”

**The handler program**

At the center of this architecture is the object handler. It is specific to the object itself, carrying with it all the methods necessary to access the information by both human and other systems.

The most difficult challenge in implementing an architecture such as this is the work involved in creating specific forms handlers. Since there will be typically thousands of different forms, writing specific object handlers for each one is not cost-effective. However, because each electronic form has a unique set of data elements, business rules, and routing processes, the use of generic handlers is also impractical.

The resolution of this problem for electronic forms is approached through the use of a “code-generator” program that determines the syntax of the hypertext markup associated with a particular form and automatically generates Perl code necessary to respond to data coming from the form to the server. Sometimes the appropriate response might be to send the form back to the client with the data just entered. On other occasions, the response would be to finalize the form’s data in the central database and make them visible in the approver’s in-box.

Most of the basic information necessary to generate the handler program is contained within the HTML description of a form. Additional functionality necessary to support complex objects, field checks, business rules, and routing is inserted through the use of pseudo-tags (embedded in HTML comments) and special parameters inserted in conventional tags.

**A point-and-click designer**

One of the most significant barriers to real user empowerment is a system requiring users to represent things in code as opposed to using visual tools. Even though users can be taught to code relatively simple objects, anything with a moderate or greater degree of complexity will present barriers that will diminish the overall usefulness of the forms tool, restrict the role of creating forms to computing professionals, and ultimately undermine the potential for electronic forms to actually reduce staffing requirements. In addition, while program-generated, HTML-based forms can be quite complex and interactive, the existing version of the language is very limited in its ability to completely represent them.

The solution is a point-and-click “forms cre-
ation and modification" tool that will permit users to create new forms using their HTML browsers. This tool will be server driven and completely based on the existing browsers. It will not be another workstation-based HTML browser (such as Arachnid) that is limited to a single platform and does not represent the object being created exactly as it will appear when it is used.

The point-and-click creation tool depends on the <table> feature in HTML v2.1 (currently supported in Netscape and Windows Mosaic). Everything in the form is part of a table. In this way, the page is divided up into individual cells. Empty cells are represented as asterisks on the screen (these can be made to disappear if you want to "proof" the page) that are active links to another series of forms that are used to gather the characteristics of the object you want to put on the page. The entry of rudimentary edit-check rules can also accompany the definition of particular fields.

Behind this display scheme is a database table in which each component of the page is stored. These components can be retrieved in proper sequence, making it possible to generate an HTML representation of the page.

HTML provides only for relatively simple forms components such as input fields or radio buttons. More complex components such as an expanding data grid can be supported by combining the simple field types that HTML does provide and coordinating them with the back-end handler. Within the designer, these complex objects are just another type of object in the system's overall catalog. The catalog can also include a wide variety of images, backgrounds, and custom buttons, so it will be possible to create attractive pages with this facility. Also included in the catalog can be standardized components such as a name and address or an account code, which carry with them an intricate set of value checks that conform to institutional data definitions.

The designer’s end product is an HTML file that is directly conveyed to the code generator. This file may contain some unique pseudo-tags and parameters to implement special features, but nothing is generated that will cause standardized browsers to produce errors. This interim HTML file is an important feature because it allows users to define forms through other mechanisms or code them directly if that is desired.

Access to specific forms is managed through the in-box mechanism. This functionality is made possible by a table in the central database. Each row in this table corresponds to a particular form and identifies in whose in-box a reference to the form appears. Routing is accomplished by modifying this identification as a form goes through its approval chain.

Access control and security

Controlling access to objects—forms in particular—is an essential component of the system. This is made more difficult by the stateless nature of WWW technology. There is no "session" with the host, so user IDs and passwords must be embedded within every page and verified with every transaction of the conversation between client and server.

There are two ways the system minimizes the exposure posed by embedding authentication in this way. First, a secure server is used that encrypts the data stream and prevents exposure through "sniffing." Second, the system uses a temporary password scheme that presents a unique authentication key every time there is an exchange between client and server. The key is periodically re-randomized, making it difficult to "jump in" and "steal" the virtual session.

Applying business rules

The application of enterprise business rules automatically as early as possible in a form's processing chain is a major benefit of using electronic forms.

Routing

Because the system's objects are server based, they never actually move—they are always contained within a single database management system. However, access to them

---

5 A sniffer is a network monitoring tool that can capture data packets and decode them to show protocol data.
6 X.500 is the set of ITU-T (International Telecommunication Union, Telecommunication Standardization Sector) standards covering electronic directory services such as white pages.
changes as a result of the operation of routing functions, thereby giving the impression of movement.

Like business rules, routing rules are embedded in the object handler and invoked at appropriate times by specific methods. Different routing rules must be used at different times in the routing process. Each step in the process has a series of rules associated with it that are only invoked when that step is current.

Routing happens because routing rules manipulate entries in the in-box table. Appropriately written rules can do much more than route—they can kick off events such as e-mail notifications, logging events, or transactional update processes.

Transaction pipeline

An important step in the routing chain for many objects—especially electronic forms—involves the transfer of information to a transactional system. This is handled by methods contained within the object handler that act very much like the methods that cause the information to be sent to a human user with a Web client.

The demons that convey information to transactional systems access the link table to determine when an object is ready for data transfer. This is indicated in the same way as other routing steps. When the right conditions have been met, the data are conveyed by the pipeline demon to a batch file. This file is sent to the appropriate transactional system on the University’s administrative mainframe.

Implementation

A fully functional prototype of the electronic forms system described here was written by the author in the summer of 1995. Following this, a team of programmers began implementation of a production system and testing of the first two forms began in February of 1996. Some features in the prototype did not make it to the production version, but the basic design and functionality are the same.

A pilot effort is under way to test the new system in two administrative departments. We have provided short training sessions to familiarize new users with the forms as they appear in their WWW browsers. Not only has resistance to the change among administrative staff proved to be considerably less than anticipated, users appear to be approaching the new system with enthusiasm.

One of the lessons learned so far is the importance of focusing on features that will make data entry easier. This includes the ability to support templates specific to a particular group of users and deliver blank forms with certain fields already filled in with values appropriate to them. Another feature frequently requested is the ability to “repeat” values from one form to another.

At the approver level, the interest is in sophisticated routing. What people at this level want is for forms bearing information that meets particular conditions to be routed in a particular way. For example, all expenditure requests for a particular type of commodity might need to be routed to a particular approver for special examination.

Another way electronic forms are adding value involves sophisticated data verification. With the current paper-based system, a fairly high level of resources is expended resolving bad data that come into the system. This can include things as mundane as bad account codes or accounts charged to by inappropriate submitters. Catching this kind of error at the moment of submission rather than later in the approval chain will enable the University to save a considerable amount of money and reduce bureaucracy.

Conclusion

The architecture outlined here takes advantage of Web client capabilities to facilitate reengineering of business systems by reducing their dependence upon labor-intensive paper-based processes. It does so by relying upon client software that is rapidly becoming a standard presence in every network-attached personal computer workstation. It is standards-based and highly modular in design, affording the maximum opportunity to take advantage of new technology as it appears. Finally, by using it we hope to deploy a secure, highly robust electronic forms system at an enterprise level with a minimum of system support costs.

Acknowledgements:

The author wishes to acknowledge the contributions of Dr. Polley A. McClure, Dave Saunders, Mike Jewell, Candace Graves, Debbie Luzynski, and the Wookie to the work described in this article.

This article was adapted by the author from a presentation he made at the September 1995 Monterey Conference, “Higher Education and the NII: From Vision to Reality,” which was included in the Monterey Conference Proceedings. These proceedings are available from Educom (202-872-4200).

Footnoted definitions of terms used in the article are adapted from the Free On-line Dictionary of Computing, located at: http://wombat.doc.ic.ac.uk/cgi-bin/foldoc/Free+On-line+Dictionary

Footnote 7: A demon is a program or part of a program that is not invoked explicitly, but that lies dormant waiting for some condition(s) to occur.