



# INTERFACE-PARASITE GATEWAYS

Robert A. Barta, Manfred Hauswirth



## Abstract

*Apart from challenging new possibilities in resource discovery and retrieval, WWW gives the opportunity to provide legacy applications with a new look and feel and to add functionality without having to change the software itself. We describe problems and possible solutions in the area of connecting legacy software based on terminal sessions to WWW, especially focusing on the problems that arise from adapting stateful interactive services to the stateless HTTP. As a case study we present a WWW gateway to a stateful legacy application (Austrian national academic OPAC) by encapsulating it within a CGI conformant gateway. **Keywords:** World Wide Web, gateways, legacy software, stateless and stateful, interactive sessions, online public access catalogs (OPAC)*

## Introduction

Providing access to nonHTTP-based information systems through the World Wide Web [6] implies that two technical problems have to be tackled. First, WWW requests have to be transformed into queries for the information system—say a database—and, second, query results have to be converted into HTML [5]. There are numerous generic gateways (e.g., [15]) and programming language extensions (e.g., [20]) available for the skilled engineer with which prototypes or even complete applications can be built.

The trouble starts if no application programming interface (API) for the information or database system in question is available, or—similarly serious—no reliable documentation for the system exists. Applications of this kind are often euphemistically referred to as *legacy applications*. Most of them originated many years ago, providing access for querying and maintenance only through interactive, character-based terminal sessions (telnet, 3270) or dedicated user interfaces.

The standard way to “integrate” session-oriented resources into WWW is by providing a telnet URL [4]. A rich list of telnet-accessible resources can be found in [19]. Clicking onto such a link will start up a terminal emulation program at the client site. The user then has to navigate according to the application’s interface philosophy which

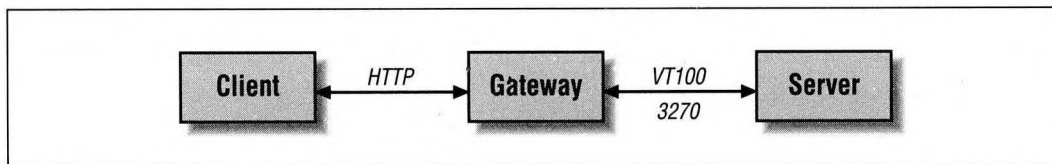
usually does not follow WWW’s user interface paradigm. Even worse, most applications expect the terminal emulation to be perfectly configured, forcing the user to use special function keys or escape sequences.

A more seamless integration can be done by writing CGI [14] gateways which encapsulate these applications. Thus users just have to deal with the usual WWW point-and-click GUI paradigm. Carefully designed gateways will also result in a much simpler user interface compared to the original (e.g., [9]).

In the following sections we first describe interface-parasite gateways, an approach we successfully used for the integration of BIBOS, Austria’s national academic OPAC. We then argue for a complete integration of gateways into existing information infrastructures. Next, we address the issue of statefulness, which is crucial in the design of WWW gateways, and present our WWW-BIBOS gateway as a case study. Finally, we describe how implementation of interface-parasite gateways can be supported by tools.

## Interface Parasites

In the absence of an API interface to an existing system, the only reference is the user interface itself. Our approach is based on attaching to tex-



**Figure 1:** Gateway as interface parasite

tual, character-stream-oriented user interfaces with (command) line or page logic.

Providing a seamless service integration into WWW means hiding the *terminal emulation feeling* from the user. This implies that a gateway has to run the terminal emulation on behalf of the client (Figure 1).

The gateway's task is to behave as a *virtual user*. It has to translate HTTP requests into (sequences of) keystrokes that are sent to the terminal emulation. When the application has reacted by showing a new *virtual screen*, the gateway has to analyze it and extract any relevant information. It does so by locally interpreting the data and control stream coming from the application. In the usual case of character-based terminal emulations this necessitates running a telnet or 3270 protocol automaton and interpreting terminal control sequences (e.g., that of VT100). Depending on the type of gateway, relevant information is hypertexted and forwarded to the client.

Detection of information is the key problem here. Any information on a user interface is first converted into an external representation (strings for textual interfaces). Usually semantical information is lost during this process. It is not possible to reverse this process in general. Often information is packed into a few lines because of space limitations on the display. This makes automated detection of information complicated or even impossible. One has to exploit position information which may give a clue to the meaning of a string or rely on heuristics.

Another problem concerns page-oriented interfaces. The end of a page has to be detected before the analysis process can start. In telnet-

based services transmission of characters usually just stops at the end of the page. While this is obvious to the human client, it is not for an automated one. In order to avoid painful timeouts, the character stream must be analyzed to identify discriminating character sequences which give a clue to an end of the page. Sometimes heuristics have to be applied here, too.

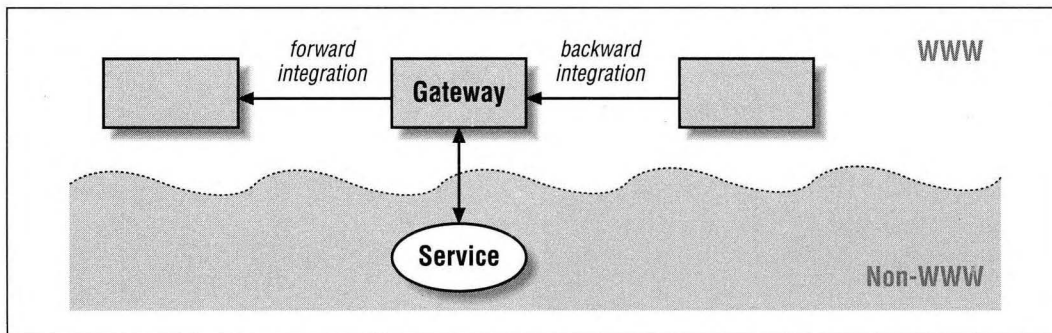
From the above it is evident that interface-parasite gateways are vulnerable to any kind of layout modifications of the user interface. These, however, are by definition not likely for legacy applications.

## Forward and Backward Integration

Providing gateways between WWW and other resources not only allows people to use this service without being bothered by its specialities. It also allows integration with other information systems (Figure 2).

Given a particular URL, the gateway will produce a document on the fly. *Backward integration* means that these virtual documents must have a systematically reproducible URL. Only then can another Web service make more use of the gateway than merely pointing to the gateway's start page. Examples of such gateways are those into X.500 or whois.

*Forward integration* conveys the approach where the gateway itself enriches its output by providing links to other relevant Web resources. A gateway to a staff database, for instance, could not only return the department and the telephone number of the sought-after staff member



**Figure 2:** Forward and backward integration

but could also generate a *mailto* link, say *commonName.surName@organization.country* supposing such an email address is valid.

The effort to maintain a small database of links to be included is often negligible in comparison to the gain of integration.

## Stateful vs Stateless Gateways

WWW is based on the stateless HTTP protocol [23]. Using it as GUI for stateless (request-reply) applications, like many C/S applications, works perfectly well. For stateful (interactive) services state information has to be kept and maintained for a certain period of time. The decision what and where state information is stored is crucial for the design [16].

### Stateful Gateways

Efficiency and security considerations may prevent the use of a simple stateless gateway approach as described below. In the OMNIS/Myriad retrieval engine [8], for example, login/logout is very resource-consuming, making it necessary to use a stateful WWW gateway with dedicated, permanently connected server processes.

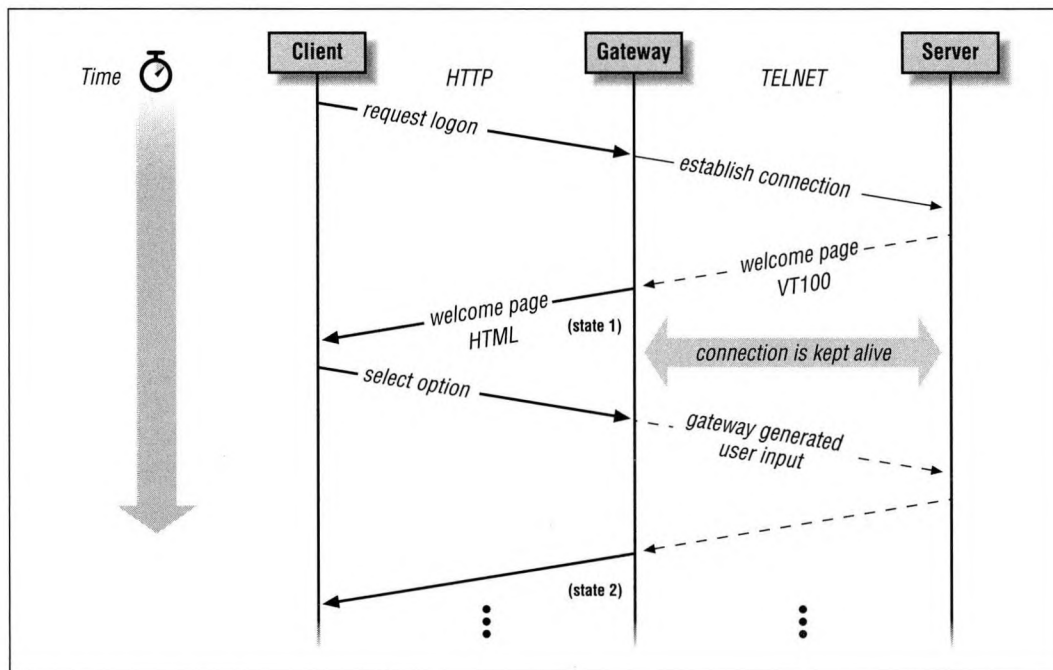
Stateful gateways overcome such problems at the expense of having to maintain state information within the gateway. Followup HTTP requests are

related by gateway-maintained state information [16].

Usually a permanently running gateway process keeps up a connection to the server on one side (stateful) and serves successive requests on the HTTP client side. Every request is translated relatively to the gateway's state (Figure 3).

Translation servers [17] work according to this interaction pattern. Though flexible and fast, this approach suffers from a state bookkeeping problem. During a session the history of all requests has to be memorized to provide a stateful connection to the server. In the presence of client caches, which all state-of-the-art browsers use, this management of context information can be quite complicated; whenever the user decides to go back to a previous page, the client will not contact the gateway, but use the cached information instead. New requests issued from previous pages (i.e., previous client states) may confuse the gateway. It may even be forbidden in certain application states, even if no cache is used. Recovery procedures for the gateway will be complex, as some kind of request unification may have to be applied or even complete playback of the whole session may be necessary.

A similar problem arises when the client interrupts an HTTP request. State may become inconsistent depending on the time the interrupt occurs and the gateway and service semantics. Recovery is hard or impossible in this case.



**Figure 3:** Stateful gateways

The gateway has to take into account existing timeouts of the session-oriented service and coordinate these with its own (e.g., [17, 8]). This problem is further complicated by unstable client software (lost connections).

Many stateful services have restrictions for concurrent access, be it license agreements or performance problems to necessitate a user limit. Stateful gateways may deny access when the user limit is reached. This may lead to request starvation. In order to cope with this problem, services will try to prevent long inactive sessions. When a user waits too long for a followup HTTP request, the connection between gateway and server will time out. On issuing the next request the gateway either reacts with an error message or has to replay the whole session up to the last state, which might be rather time- and resource-consuming.

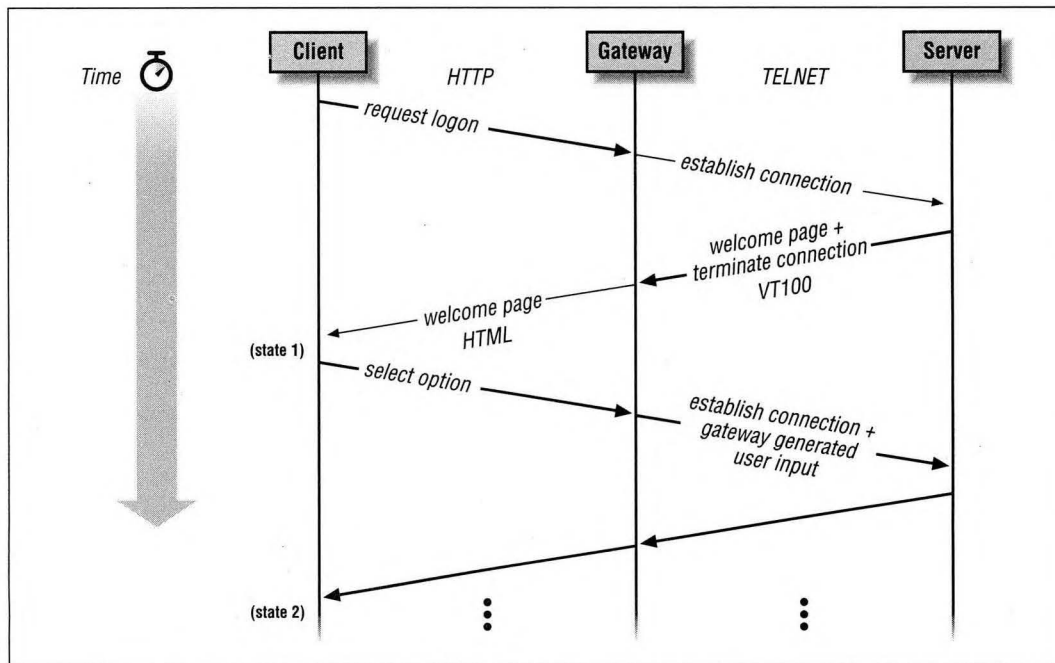
### Stateless Gateways

Stateless gateways are the simpler type of gateway for stateful services. State information that is necessary to relate HTTP requests and replies is generated by the gateway and stored at the client either within URLs (see [18] for an example) or inside HTML code as hidden fields in forms. Thus stateless gateways make it possible to overcome the possible inconsistency problem of stateful gateways.

There are two extreme approaches:

The *bridge* mirrors the logical user interface structure on the server (e.g., the sequence of terminal pages a user has to navigate through) into WWW pages. User interaction is translated one by one into HTTP requests.

Since the communication pattern itself is stateless, a followup request causes the whole interaction to be replayed up to that context, which has already been built up between the client and the



**Figure 4:** The bridge

service. This implies running through all previous states to build up this context before issuing the new request (Figure 4).

With a similar technique clients themselves work on FTP links. Here the client itself uses the link information to start an FTP session and navigate to the requested directory or file. Afterwards the FTP connection is closed and has to be reopened for followup FTP requests.

This type of gateway is mainly feasible for inherently stateful services like [18] or services that have some kind of “collect and submit” semantics; e.g., consider a shopping service where the user “collects” items and when finished submits a “buy” request. While collecting, the user can easily go back to any previous state (in case of mistakes), pause as long as wanted, and carry on later.

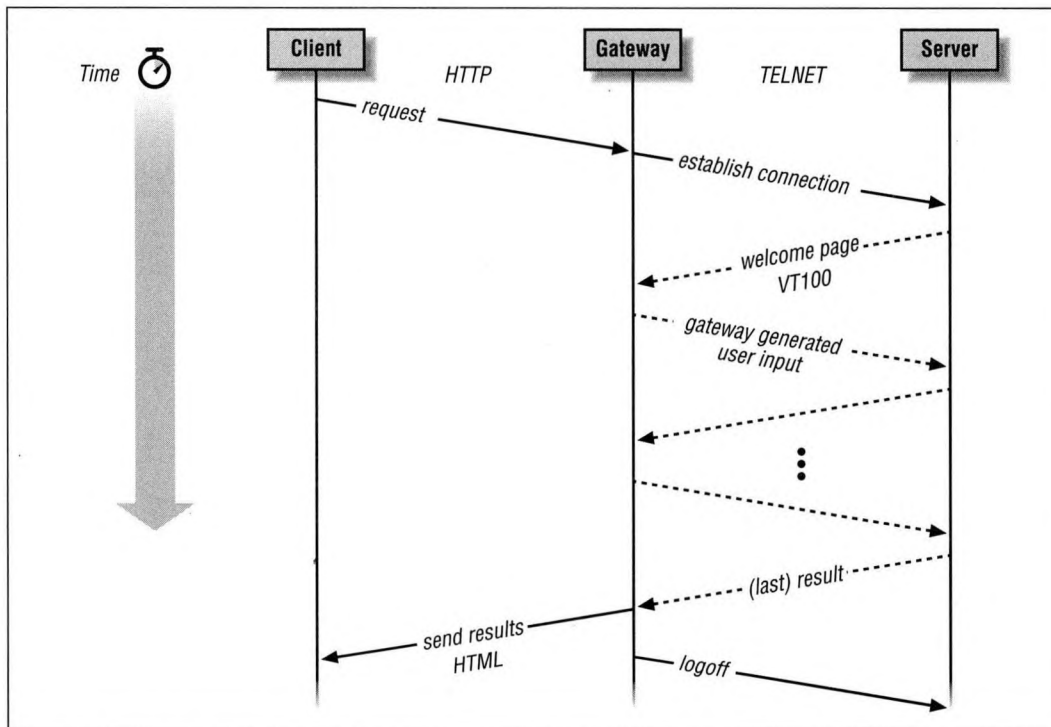
However, this solution also has some significant drawbacks: state information can quickly become rather large and due to replaying of sessions

computing resources are wasted. Additionally, since state information is visible to the user, there is no protection against modification of this information by the user.

*One-shot gateways* transform the application semantics itself and not only the syntactic server information. This is accomplished by a request/reply interaction hiding intermediary states (pages) from the user.

This means that the user specifies the complete input information with a single HTML form [7]. After submitting it, the gateway runs a complete session on behalf of the user and returns the relevant information in hypertext representation (Figure 5). This frees the gateway from keeping connections open and provides a higher degree of scheduling freedom. As a byproduct this method offers inconsistency transparency [3], since no state information has to be kept between successive requests.





**Figure 5:** One-shot gateways

This approach also facilitates a meaningful, seamless integration into WWW. Result information can be collected throughout the session and can be combined, converted, corrected, and enhanced at will and presented in a WWW conformant, hypertext way, not merely mirroring the legacy original.

This implies that the prevailing application has to be analyzed thoroughly. This approach also assumes that all necessary input is provided by the user with a form, redefining the service into a request/reply application. If this is not possible, more sophisticated gateways have to be used. Nevertheless, it seems that a considerably high percentage of applications can be transformed into one-shot applications.

## Case Study: WWW-BIBOS Gateway

BIBOS is the Austrian national academic online public access catalog (OPAC) and encompasses almost all of Austria's academic libraries. It is a huge database running on a mainframe for several years now. Before the existence of the gateway, BIBOS only offered terminal emulation access [*telnet://opac.univie.ac.at*]. Some major drawbacks of this access method are:

- Users (especially students) are reluctant to learn how to search and navigate through BIBOS.
- Terminal emulations have to be configured well because the use of special function keys is mandatory.

**WWW <-> BIBOS Gateway V2.2**







A [more verbose form](#) and a [more powerful one](#) is available. We have also a [newsletter](#) in which you find current information about this gateway.

Author

Title

Pressing **Search** shall return at most  **compact**

Library

*Monfred Hauswirth, Robert Barta*

**Figure 6:** Gateway query form

- Several pages have to be passed and filled out before the actual search form is reached. The same applies to logging out.
- Important functionalities are unnecessarily complicated to use; e.g., to get detailed information on books, users have to perform a query to get a list of matches first. Only then can one step through the search result and request detailed book information.
- BIBOS is full of acronyms and codes, only understandable to the professional librarian.

The database itself, however, offers exhaustive access to scientific publications. Since there was no BIBOS documentation available to us, we decided to prototype an interface-parasite gateway using the one-shot approach described earlier.

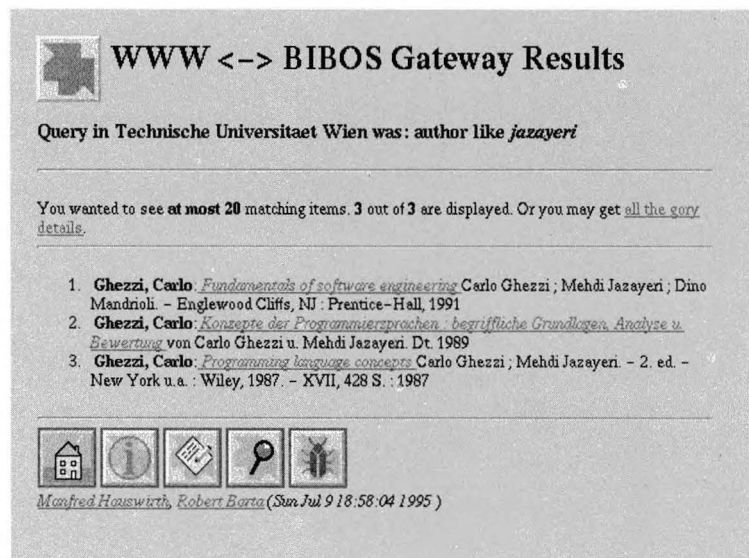
Initially the user is confronted with one out of three possible query forms: a form with all available input fields [<http://bibgate.univie.ac.at/>], the same form with descriptions for the fields [<http://bibgate.univie.ac.at/BIBOS-2/Search-verbose.html>], or a concise one [<http://bibgate.univie.ac.at/BIBOS-2e/Form.pl>] (Figure 6), only con-

taining the most important fields. All types of forms are linked to each other. Additional information (help, feedback, usage, deficiencies) is available via buttons at the bottom of each query page.

Having filled out the query specification, the user has to submit the form. On return, he/she will receive a list of matches. Every match is displayed in a compact or detailed way, depending on the user's selection in the query form. Figure 7 shows a compact query result.

Clicking on a specific match will repeat the query and will return a more detailed description. If the user wants detailed descriptions for all books there is a *gory details* link at the top of the result page.

To get in contact with our customers we offer a feedback form to them. Initially this was only a simple form where users could enter their freely formulated opinions. As we received only a few, mostly imprecise statements, we decided to offer an exhaustive questionnaire. We now receive lots of feedback with more meaningful bug descriptions and suggestions for improvements.



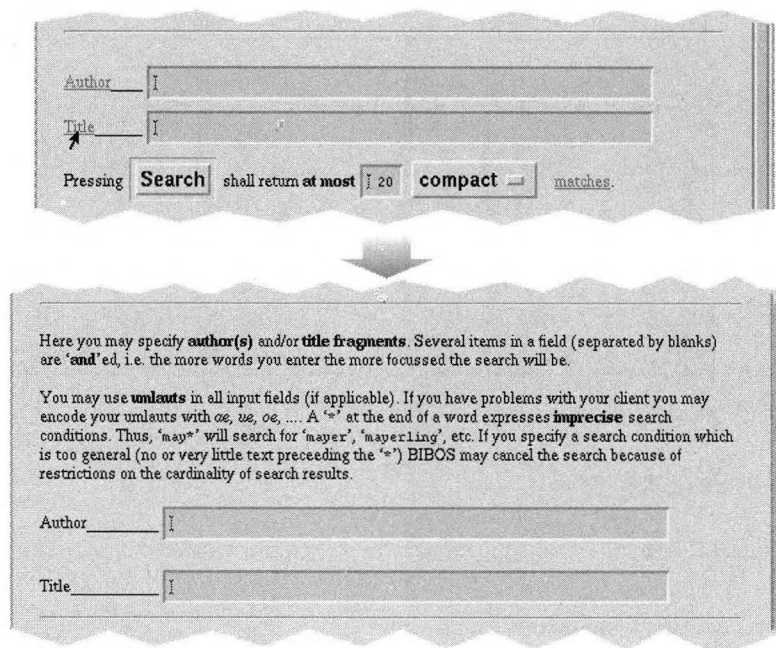
**Figure 7:** Query results

### Extending the Functionality

The gateway offers a wealth of additional functionality to the user which was not present before.

- Detailed search results can now be requested directly. Also, the user may now restrict the maximum number of matches returned, to have better response times. Future gateway versions will provide a taggable list of compact results, where the user can select the matches that will be displayed in detail.
- In order to support sporadic users we supply a simple context-sensitive help facility. By clicking on the label of a query field a verbose query form is displayed at corresponding page position (Figure 8). The user then can go back to the initial form (where parts of a query might already have been specified) or use the verbose form.
- Some entry fields in the original BIBOS interface require the user to enter special names and codes, e.g., library selection or the type of medium to look for. In the gateway these fields are offered through selection lists.
- We tried to integrate the existing information infrastructure into the gateway (forward integration). If, for instance, a particular book is located in a department and not in a library, we refer into an information system about departments and their staff (currently works only for our university). Otherwise we link to the library information service.
- While BIBOS allows a keyword search, the list of valid keywords is stored separately. The gateway provides access to that list via a link.
- Additionally, we analyze shelf codes and enhance these alpha-numerical codes with a more meaningful, user-friendly text and hyperlinks, if an appropriate information system exists (currently works only for our university).
- For backward integration the gateway now allows URLs to point to BIBOS references. These BIBOS links can easily be embedded into other HTML documents. Some Austrian





**Figure 8:** Context-sensitive help

universities have already included publication lists of their staff into their information systems by adding hyperlinks into the WWW-BIBOS gateway. Instead of having to fill out forms and submitting them, corresponding hyperlinks are generated automatically with appropriate parameters and link activations result in a query to BIBOS.

Other improvements simply result from using state-of-the-art WWW browser. Users now can go back to any previous page at any time using client caches. This is not possible in the page logic of BIBOS itself.

Due to BIBOS' terminal interface only a limited number of items can be displayed per output page. The user has to jump back and forth. Query results cannot be printed from inside BIBOS. The gateway, however, delivers all matches in a single HTML page which can also be printed readily.

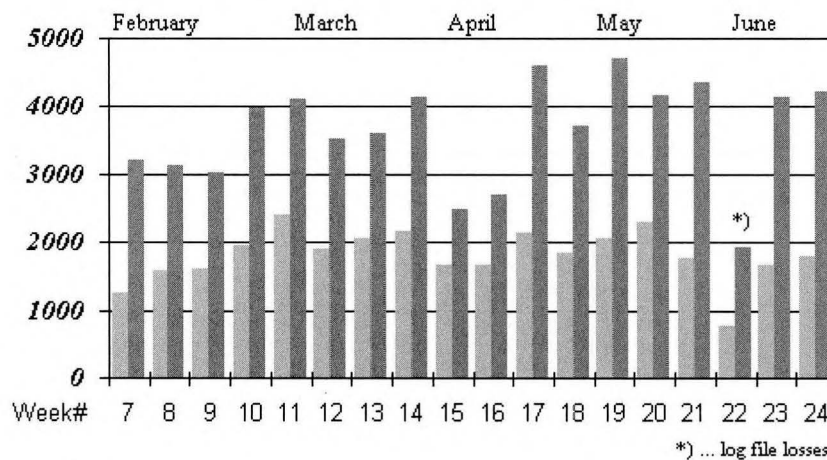
## Implementation Problems

In addition to the general issues of stateless interface-parasite gateways we had to cope with other problems when implementing the gateway.

The most serious and basic problems we encountered were the lack of an API and any documentation of the interface. We initially had to analyze BIBOS' output behavior which we did in a long and painful trial-and-error process.

Since we did not want to implement a full 3270 protocol automaton, we decided to use tn3270, which offers a telnet-based 3270 emulation. Our gateway then analyzes VT100 control sequences to keep a local virtual screen.

The virtual screen is necessary because the BIBOS interface returns data whose semantics derive from its position on the screen. Additionally, there exists no standard format for the reference information returned by BIBOS. The gateway has to guess which parts of a reference



**Figure 9:** BIBOS gateway usage

denote author, title, bibliographic and administrative information.

There are several situations when the gateway cannot precisely detect whether a page has been transferred completely or not. This end-of-page detection problem could only be solved by using appropriate heuristics.

Finally we had to cope with a couple of special cases the existing BIBOS interface exhibits. So, for instance, a query process works slightly different when searching in all libraries at once rather than in a single one. Also BIBOS returns detailed information instead of compact references when only one match is found for a query. These nonorthogonalities do not make sense in a WWW context, so the gateway hides them from the user.

### Performance Considerations

Introducing such a gateway clearly simplifies the handling for the end user. Still, one should also consider the effects on the involved hosts and the traffic produced by such a service, especially when it gains widespread popularity. Figure 9 depicts access statistics of our gateway. The smaller bars represent the number of query-form fetches; the taller ones are actual search requests.

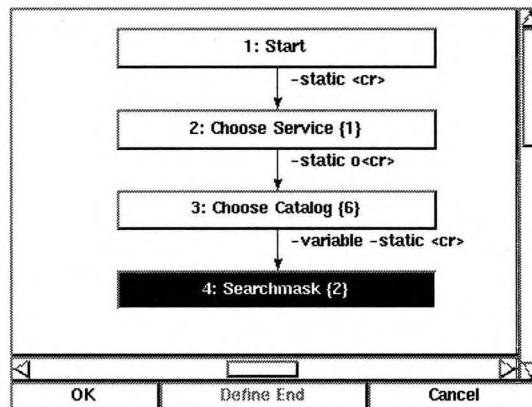
In pregateway times people were using telnet and, less often, 3270 emulations. The former is quite costly in bandwidth terms, since every typed character results in two packets on the network. Result pages may be fragmented, adding to the network cost. This problem is less serious with 3270 since it transmits on a page-by-page basis.

Our gateway, in turn, receives HTTP requests, uses 3270 as terminal emulation protocol, and ships back HTTP packets. This—especially over long distances—is cheaper than having a VT100-based telnet session.

This traffic reduction is attained at the cost of additional processing load on the gateway and on the database host. Currently, for every request a new 3270 connection has to be established. Since the gateway is stateless, the database host may be forced to repeat queries already performed. Nevertheless, it does not have to keep open long-lived sessions, thus giving it more scheduling freedom.

### Planned Improvements

The gateway has left its prototype phase and is about to be institutionalized. This necessitates several procurements, such as providing a better online documentation and adding search demos.



**Figure 10:** Interface logic

Also, more sophisticated management tools have to be considered.

The most important goal, however, is an increase in performance. As already mentioned, the connection establishment is responsible for a good deal of the delay. To overcome this, we will run permanent terminal-emulation sessions. Particular search requests will be assigned by a global scheduler to one of these sessions. This will also lead to a more modular gateway architecture.

Concerning the functionality, we have to set up a more manageable, i.e., scalable scheme to integrate existing information systems. The idea is to include pointers to information systems that belong to the organization which hosts a book, or, if available, to link to an electronic version of the document. Furthermore comprehensive analysis of classification and shelf codes will be performed (currently this analysis is done only for our local university library). Their syntax and meaning depends on the sublibrary, making necessary an incremental process depending on the support we receive from the librarians.

## Spin-off Projects

Our experiences with the gateway have encouraged us to benchmark the approach with other services. This resulted in prototypes for the Aus-

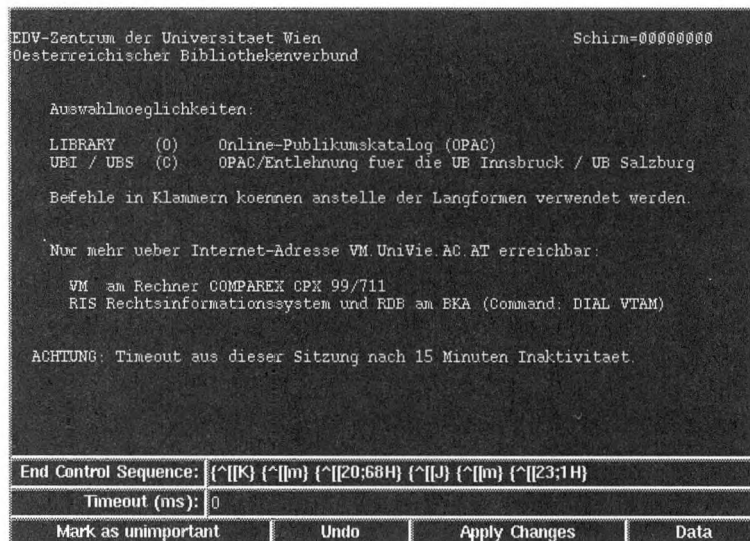
trian electronic telephone book (not publicly available) and for the Austrian stock exchange (service is cancelled due to licensing problems).

In general, during a session with an interactive service a user is confronted with a couple of pages. Each single page requires the user to input some commands or fill out a form. All possible pages build a network with the pages as nodes and commands as edges. Currently we are working on a tool which will support the designer when analyzing such a network.

## Interface Logic

With the help of the tool the designer navigates through the pages. The tool memorizes pages already encountered and unifies those pages—with the help of the designer—which have to be regarded to be the same. During this process the tool learns about the significant and nonsignificant parts of a page.

In the same way the commands (edges) that link the individual pages are learned. In particular, the tool gains knowledge of which parts of a command are constant and which are variable (Figure 10).



**Figure 11:** Query session

## Page Information

For every page the designer has to mark the positions on the screen where relevant information is expected. In this phase the tool learns about the semantics of a particular piece of information.

## Query Session

Now the designer has to define a path through the network of pages. Along with that he has to specify how input parameters have to be mapped to the variable parts of commands (Figure 11).

## Backend Generation

In this phase a code skeleton (we plan to use perl [22]) for running a query session is generated. Afterwards the designer has to supply code that defines how the collected information is presented to the client.

The tool will free the designer from digging into character codes and will help to adapt to small modifications in the interface logic.

## Related Work

Considerable effort has been spent in many recent projects to integrate existing databases and legacy systems into WWW. Different application domains and requirements led to a high variety of solutions.

After a proof-of-concept gateway succeeded, the MORE [9] project team decided to completely redesign the underlying database. Switching from a stateful database environment to a stateless WWW interface environment transformed the original complex multifunction windows interface into a rather simplistic single function/single action interface.

Special attention to the user interface design area of gateways is paid in the OMNIWeb system [13]. Typical user sessions are analyzed to meet user requirements and a context-sensitive help facility is added.

Instead of using specialized gateways to access legacy data, the CUBAI project [2] promotes (bijective) conversion of such data into a common format, facilitating integration and interchange of heterogenous data. The benefit of

needing only one WWW gateway to access the common data, however, is achieved at the expense of high conversion costs.

The OmniPort system [10] tries to integrate legacy data by bidirectionally translating the native search capabilities into a standard language. Thus consistent access to multiple information sources is offered regardless of particular access methods. It supplies new uniform search capabilities using WWW as front-end for easy and everywhere availability. A similar approach is used in Willow [21], which uses the architectural paradigm of database drivers to communicate to various databases in a uniform way. A driver for Z39.50 [1], a standard that defines a uniform procedure to query information resources, is supplied.

Other approaches try to cope with the problem of generating gateways by describing the original user interface and thus deriving an executable gateway. Expect [12] allows a definition of dialogs with interactive programs by programmatically characterizing interactions.

IDLE [17] is a description language for interactions with telnet-based, character-stream-oriented interfaces. IDLE programs are interpreted on the fly by a so-called translation server [16], thus providing the intended WWW-gateway functionality. Problems of gatewaying between stateless and stateful protocols are addressed in [16].

## Conclusion

A considerable number of useful services is provided by so-called legacy systems. This does not necessarily mean bad software, but it indicates that its production standards make it difficult to modify it in respect to user interface techniques and/or functionality [11]. Especially code for the user interface is tightly embedded into the application's code and cannot be upgraded easily to modern user expectations.

Interface-parasite gateways are a cheap and—when carefully designed—an effective way to integrate an existing service into a global infor-

mation infrastructure like the World Wide Web. Their employment provides a new look and feel to an application and prolongs the application's lifetime. ■

## References

1. ANSI/NISO, *Information Retrieval (Z39.50): Application Service Definition and Protocol Specification*, technical report Z39.50-1995, May 1995. <ftp://ftp.loc.gov/pub/z3950/itr/itr1-41.ps>
2. A. Balestra and M. Ferrucci, *Accessing libraries in a uniform way: the CUBAI project*, contribution to the workshop "Offering the same information via multiple services," First International Conference on the World Wide Web, Geneva, May 1994. <http://www.oat.ts.astro.it/www94/paper.html>
3. R. A. Barta, *Formal specification of distributed systems - A discrete space-time logic*, PhD thesis, Vienna University of Technology, 1995. <http://www.infosys.tuwien.ac.at/Staff/rho/Dissertation/Dissertation.html>
4. T. Berners-Lee, *Uniform Resource Locators*, 1992. <http://www.w3.org/hypertext/WWW/Addressing/Addressing.html>
5. T. Berners-Lee, *HyperText Markup Language (HTML)*, 1993. <http://www.w3.org/hypertext/WWW/MarkUp/MarkUp.html>
6. T. Berners-Lee, R. Cailliau, A. Loutonen, H. F. Nielsen, and A. Secret, *The World Wide Web, Communications of the ACM*, 37(8), August 1994.
7. T. Berners-Lee and D. Connolly, *Hypertext Markup Language - 2.0*, World Wide Web Consortium, June 1995. [http://www.w3.org/hypertext/WWW/MarkUp/html-spec/html-spec\\_toc.html](http://www.w3.org/hypertext/WWW/MarkUp/html-spec/html-spec_toc.html)
8. A. Clausnitzer, P. Vogel, and S. Wiesener, *A WWW interface to the OMNIS/Myriad literature retrieval engine*, Third International World Wide Web Conference (Darmstadt, Germany, 10-14 April 1995), *Computer Networks and ISDN Systems*, 27(6):1017-1026, Amsterdam, North-Holland, Elsevier, April 1995. <http://www.igd.fhg.de/www/www95/papers/65/omnis-www95/omnis-www.html>
9. D. Eichmann, T. McGregor, and D. Danley, *Integrating Structured Databases Into the Web: The MORE System*, First International Conference on the World Wide Web (Geneva, May 1994), *Computer Networks and ISDN Systems*, 27(2):281-288, 1994. <http://www1.cern.ch/PapersWWW94/more.ps>
10. S. G. Ford and R. C. Stern, *OmniPort: Integrating Legacy Data into the WWW*, Second International Conference on the World Wide Web (Chicago,



- October 1994). <http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/CorInfSys/stern/stern-ford.html>
11. H. Gall, R. Klösch, and R. Mittermeier, *Architectural Transformation of Legacy Systems*, ICSE-17, Workshop on Program Transformation for Software Evolution (Seattle, USA, April 1995), April 1995.
  12. D. Libes, *Expect - programmed dialogue with interactive programs*, National Institute of Standards and Technology, July 1994.
  13. G. J. Mathews and S. S. Towheed, *NSSDC OMNI-Web: The First Space Physics WWW-Based Data Browsing and Retrieval System*, Third International Conference on the World Wide Web (Darmstadt, Germany, April 1995), *Computer Networks and ISDN Systems*, 27(6):801-808, Amsterdam, North-Holland, Elsevier, April 1995. [http://www.igd.fhg.de/www/www95/papers/68/omniweb\\_paper.html](http://www.igd.fhg.de/www/www95/papers/68/omniweb_paper.html)
  14. R. McCool, *The Common Gateway Interface*, 1994. <http://boohoo.ncsa.uiuc.edu/docs/cgi/overview.html>
  15. J. Ng, *GSQL - a Mosaic-SQL gateway*, NCSA, December 1993. <http://www.ncsa.uiuc.edu/SDG/People/jason/pub/gsql/startthere.html>
  16. L. Perrochon, *Translation servers: gateways between stateless and stateful information systems*, Network Services Conference (London, November 1994), November 1994. <ftp://ftp.inf.ethz.ch/doc/papers/is/ea/nsc94.html>
  17. L. Perrochon and R. Fischer, *IDLE: Unified W3-access to interactive information servers*, Third International Conference on the World Wide Web (Darmstadt, Germany, 10-14 April 1995), *Computer Networks and ISDN Systems*, 27(6):927-938, North-Holland, Amsterdam, Elsevier, April 1995. <ftp://ftp.inf.ethz.ch/doc/papers/is/ea/www95.html>
  18. S. Putz, *Interactive Information Services Using World Wide Web Hypertext*, First International Conference on the World Wide Web (Geneva, May 1994), *Computer Networks and ISDN Systems*, 27(2):273-280, 1994. <http://www1.cern.ch/PapersWWW94/putz.ps>
  19. P. Scott, *HYTELNET*, Northern Lights Internet Solutions, Saskatoon, Sask, Canada, 1995. <http://galaxy.einet.net/bytelnet/START.TXT.html>
  20. K. Stock, *Oraperl*, 1992. <ftp://ftp.switch.ch/software/sources/perl/db/perl4/oraperl/>
  21. University of Washington, *Washington Information Looker-upper Layered Over Windows (Willow)*, 1994. <http://www.washington.edu:1180/willow/home.html>
  22. L. Wall and R. L. Schwartz, *Programming perl*, Cambridge, MA, O'Reilly & Associates, 1992.
  23. World Wide Web Consortium, *Hypertext Transfer Protocol (HTTP)*, 1995. <http://www.w3.org/hypertext/WWW/Protocols/HTTP/HTTP2.html>

## About the Authors

### Robert A. Barta

[<http://www.infosys.tuwien.ac.at/Staff/rbo/>]  
EUnet Austria

Robert A. Barta is information manager at EUnet Austria [<http://www.eunet.co.at/>]. He received his M.Sc. in Computer Science in 1991 and a Ph.D. in Computer Science in 1995 from Vienna University of Technology. His current research interests include formal specification techniques, resource discovery, and distributed-information management. When this work was done Dr. Barta was at the Distributed Systems Department [<http://www.infosys.tuwien.ac.at/>], Information Systems Institute at the Vienna University of Technology. [rbo@eunet.co.at](mailto:rbo@eunet.co.at)

### Manfred Hauswirth

[<http://www.infosys.tuwien.ac.at/Staff/pooh/>]

Vienna University of Technology  
Manfred Hauswirth is a Ph.D. student at the Distributed Systems Department [<http://www.infosys.tuwien.ac.at/>], Information Systems Institute at the Vienna University of Technology. He received his M.Sc. in Computer Science in 1994 from Vienna University of Technology. His research interests include distributed-information management, distributed multimedia systems, resource discovery, and user interfaces.

[M.Hauswirth@infosys.tuwien.ac.at](mailto:M.Hauswirth@infosys.tuwien.ac.at)