WebSubmit: Web-based Applications with Tcl

Ryan P. McCormack, John E. Koontz, Judith Devaney
Scientific Applications Support Project
Information Technology Laboratory
National Institute of Standards and Technology

Abstract

WebSubmit is a Web-based framework providing seamless access to applications on a collection of heterogeneous computing systems. Strong authentication methods allow users to execute tasks on remote systems as if they were directly connected. Each application is presented as an HTML form; this form is filled out and submitted to the server, which then processes the request and executes the desired tasks (as the current user) on the specified remote system. The software is implemented as a collection of Tcl CGI scripts and a large body of Tcl support code. Tcl packages and namespaces are used to encapsulate functionally-distinct bodies of code, and create a system that is flexible, extensible, and modular.

1 Introduction

Experience has shown many users that the Web can be a lot like television: there may be 500,000 channels, but how many do you actually want to watch, and in the case of the Web, how long are you willing to wait to watch them? This view is centered on the notion of the Web as an aesthetically-pleasing and often mesmerizing information resource. While the role of the Web as an unparalleled data resource is not to be underestimated, it has potential beyond that of the biggest library ever constructed. Some developers and applications are attempting to harness some of this potential; we believe that **WebSubmit**,[1, 2] driven by a Tcl engine, is such an application.

The Web is traditionally used as a method for transmitting documents and images; the introduction of Java has also provided a means for distributing executable content. Additionally, restricted tasks can be performed on server machines via the Common Gateway Interface (CGI). CGI tasks have primarily been restricted to those that can be accomplished anonymously. They are executed as the server user (e.g., `nobody`) and have only the limited access privileges of this account. In most cases, this is appropriate; allowing random users to execute any command on the server system is tantamount to giving away the keys to the store. However, one would still like to provide valid, trusted users with access to resources. The novelty of WebSubmit is that it uses existing technologies to establish this trust relationship and then creates a friendly environment through which trusted users can actually use application software on the Web. This has the potential to transform the Web from a medium primarily geared towards information exchange to one in which actual computing can be done.

At present WebSubmit provides access to high-performance computing facilities at the National Institute of Standards and Technology (NIST). It was originally conceived to aid users in dealing with the complex batch submission facilities of the IBM SP2 parallel supercomputer, but it is by no means limited to this application, or, indeed, to supercomputing applications at all. It is possible to use the basic WebSubmit paradigm in any environment where users need access to abstract computing resources. In this paper, we hope to illustrate the basic structure of WebSubmit, applications within this environment, Tcl implementation and design specifics, and future directions for the software and project.

2 The WebSubmit Paradigm

WebSubmit is a framework for a secure, Web-based interface that provides access to diverse applications across a (heterogeneous) cluster of networked computers. A simple transaction model governs the way in which resources are accessed and utilized, and a security framework has been put in place to determine whether a client is allowed to perform transactions. The software itself is the combination of a collection of CGI scripts (written with the aid of Don Libes' `cgi.tcl` library [3]) and a large body of Tcl code to provide functionality for these scripts. It is modular, flexible and extensible; hooks exist for including existing CGI code and for developing and adding new applications. The code is highly portable and can be modified to suit the needs of a given site quickly and easily. The following sections provide more details on the various facets of WebSubmit.

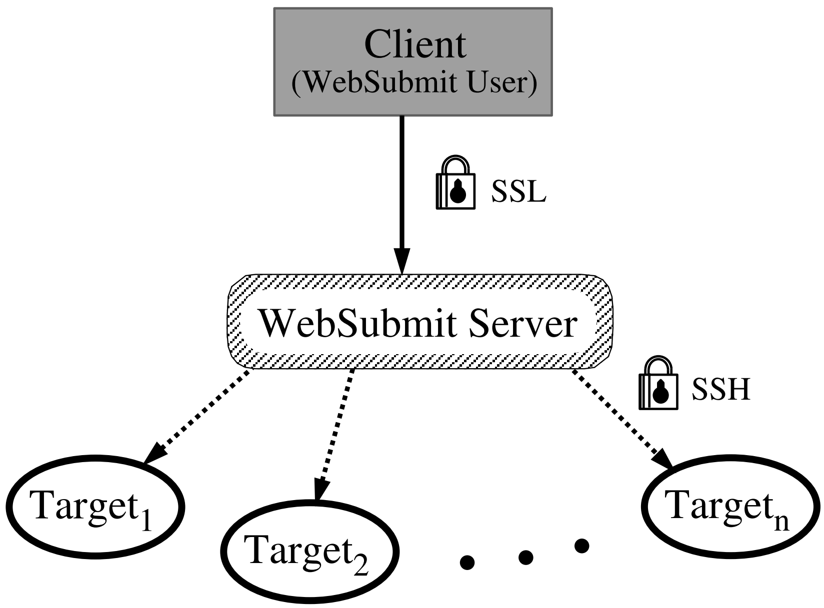

Figure 1: The basic WebSubmit architecture. A client contacts the WebSubmit server using a Web browser, which then forwards requests to any number of target computing resources.

2.1 Transaction Model

WebSubmit is designed as a tool to help users accomplish specific tasks on one or more computer systems. Each of these tasks is achieved within a basic three-party transaction model (see Figure 1): (1) the *client* that is attempting to access resources, (2) the WebSubmit *server* that formats and routes client requests, and (3) the *target* computer on which the request (or task) is processed. The client uses a Web browser to form a secure connection with the WebSubmit server's master page, then loads the *application module* for the task of interest. This module is presented as a simple HTML form that the user fills out and then submits to the WebSubmit server. The server processes the form, performs error checking on the input data, and then executes the task specified. Output from the request is then returned to the user's browser for viewing. This process is detailed in Figure 2.

In principle, any type of client system (UNIX, PC, Macintosh) could be used to connect with the WebSubmit server, provided the browser software on the client supports the Secure Sockets Layer (SSL) protocol and HTML 3.0. The WebSubmit server resides on a single UNIX machine running an HTTP server configured to provide the necessary security facilities (see [4] for more detail). The WebSubmit server is configured to interact with a group of one or more target systems (hereafter referred to as the WebSubmit *cluster*) specified by the administrator of the system. For security reasons, modules can interact only with systems within this cluster.

2.2 Authentication and Security

One of the primary concerns in a system like WebSubmit is security, with user authentication being the core issue. Indeed, this is of primary concern in seamless and metacomputing systems in general.[5, 6, 7]

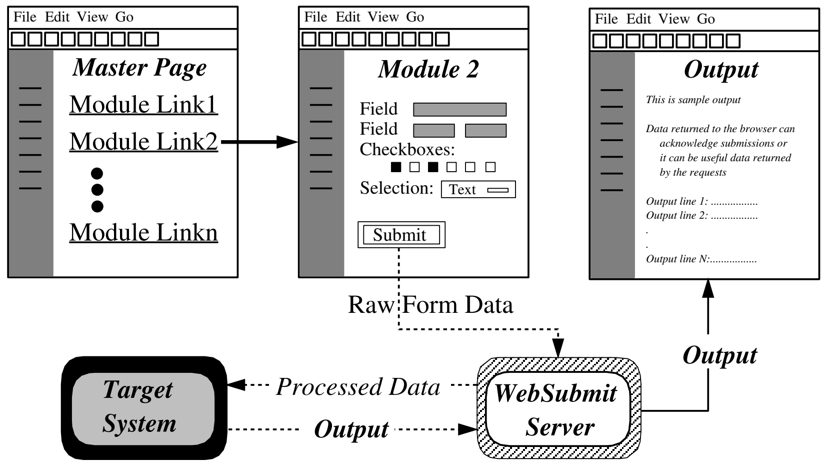

Figure 2: A simple WebSubmit transaction. The client connects to the WebSubmit master page and then selects the link for the desired application module. The module is an HTML form, which is filled out and submitted to the server. The server processes the raw form data and sends an action request to the target system. The system executes commands based on form data, then returns the output to the server, which is then forwarded back to the user's browser.

Since the user is executing commands on remote systems using the WebSubmit server as a proxy, it must be possible to establish his or her identity with certainty. Without strong authentication, a rogue hacker might impersonate a valid user and then use WebSubmit for nefarious purposes, possibly corrupting data and compromising the integrity of systems in the cluster. Most users and administrators would probably not consider this to be a feature, so a system has been implemented in WebSubmit to prevent such breaches of security. A previous version of WebSubmit [1, 2] used a combination of basic HTTP authentication (via `.htaccess`) with an SUID wrapper utility (`cgiwrap` [8]) to allow execution of CGI scripts as the required user. This architecture was not deemed adequate, since basic authentication is subject to a simple dictionary attack.[9] The new, more robust, WebSubmit security architecture works in several stages:

SSL Authentication The HTTP server running on the WebSubmit server machine is an SSL-capable [10] server configured to require client authentication to access the WebSubmit resources on this machine. Client certificates are issued by a central WebSubmit certificate authority (CA).[9]

Verification Special server code written using a public domain certificate utility package [11] extracts data from the user's certificate and establishes his or her WebSubmit user ID, name, e-mail address, and certificate issuer. This information is retrieved from a WebSubmit authentication database (`auth.db`) and validated. Only registered users are allowed access to the system.

Translation Each user's unique ID is translated into a login name on the desired target system (using auth.db).

SSH Execution Any form submitted to the system is processed and commands on target machines are executed with the appropriate login using the secure shell (SSH) or secure copy (SCP) commands.[12, 13]

This multistage security architecture presents some administrative and policy issues for the WebSubmit administrator and administrators on machines in the cluster. Without this secure infrastructure, the software would constitute a threat to the entire cluster. With it, users and administrators can sleep easily knowing that their data and systems are safe (at least from people trying to misuse WebSubmit).

2.3 Design Objectives: Flexible and Modular

WebSubmit has been designed with flexibility and modularity in mind. The basic philosophy was to create a system that allowed for diverse sites, varied user skill levels, and ease of use.

2.3.1 Site Configuration

Clearly, different sites will have different needs in terms of the applications available to users, and this issue has been addressed. The main entry point to WebSubmit is a master page that has links to all relevant application modules, help, and configuration information. Application modules are organized hierarchically on the master page, each module under a relevant application class or heading. WebSubmit allows administrators to turn modules or application classes on and off as needed, or to add new applications, simply by changing the master page. The master page is generated automatically from a database, and adding a new module means adding a few lines to this database. Users don't have access to the site-wide databases, but they can use the configuration features of WebSubmit to make undesired applications or classes invisible.

2.3.2 Application Modules: Plug-in, Hack, or Generate

Application modules used in the WebSubmit framework can come from a combination of three sources. If a site already has a set of HTML forms and CGI processing scripts (written in any language), then these can be plugged into the WebSubmit hierarchy directly with little or no modification. Alternatively, scripts and forms can be written in a developer's favorite scripting language (Tcl, of course) and then included. Finally, a mechanism for specifying simple application forms, and for their subsequent processing, has been provided to allow development without any knowledge of scripting languages or CGI. This last approach is discussed in more detail in Section 3.

2.3.3 Modes and User Skill Levels

Another layer of interface flexibility has been added by associating a modality with each application module. In an attempt to address the needs of both experienced and infrequent users of the system, each module supports at least two modes (advanced and basic). Each mode presents users with different levels of control of the application, i.e., a varying number of elements are presented on the form and default values are chosen appropriately where elements are omitted.

2.3.4 Sessions

WebSubmit supports a notion of state that makes the software easier to use and more flexible from the user's standpoint. A session in WebSubmit consists of the set of values selected for the various form input elements for a single module. For each module, a default session exists to assign values to these elements in the absence of other information. In addition, a user can fill out any form and save a named session to be loaded at a later time, thus reducing the need for repeatedly entering the same information.

2.4 Abstract Interface Specification

One design criterion for WebSubmit was the requirement for a method to dynamically generate HTML pages and forms. In order to make the system flexible and easy to modify, it was decided that, in addition, this method should not require users and developers to know the details of Tcl. There are two obvious approaches to achieve these goals: (1) create a high-level specification language that controls the appearance of forms, or (2) specify the forms through a well-defined data structure (such as a database). In our opinion, a database system is easier to implement and simpler to use than a linguistic specification, which would require users to learn another language in the place of Tcl. Simple databases are a central feature of WebSubmit, and function not only as data repositories, but also as interface descriptions. For an example of a user interface generated by a language specification, see Ref. [14].

Information related to site configuration, application modules, form element descriptions, tasks and authentication data are contained in databases that can only be accessed by the WebSubmit administrator. User databases are maintained to control the appearance of the master page and to store session information. This common data formatting mechanism allows for easy retrieval and modification of the data, as other authors of Tcl applications have noted.[15] The basic database consists of a collection of records, each of which is a group of attributes (data fields) keyed by a unique value and delimited by colons. More details about our Tcl implementation of simple databases can be found in Section 4.3.

2.5 Current Application Modules

At present, WebSubmit has several generic application modules (applicable across all systems in the cluster), and a few modules specific to the NIST IBM SP2. Three generic modules have been built at this point: a command interface, a simple file editor, and a file transfer utility. The command interface allows users to execute commands on remote (UNIX) systems. The file editor allows users to make quick changes to ASCII files on remote systems and save them. The file transfer interface provides a way to transfer single files or collections of files (i.e., entire directory trees) between systems. General SP2 modules have been developed to submit batch jobs to the queuing system (Load Leveler) and to monitor the jobs on the system. A more specific interface has also been built for Gaussian 94 [16] (a quantum chemistry package), and additional modules are planned for the Message Passing Interface (MPI) and code compilation on different platforms. In principle, there are few constraints regarding the types of modules that can be constructed. This is largely left to the imagination of the user and interface developer communities.

3 Generation of Application Modules

Developers unfamiliar with CGI programming are often left in the lurch when it comes to developing applications for the Web. In an attempt to address this problem, a facility has been created in WebSubmit to allow the specification of forms through simple databases.

3.1 HTML Forms as Application Interfaces

It is difficult to imagine a way to specify arbitrarily complex forms using a database specification. This difficulty is not created by the complexity of input elements, but by the degrees of freedom one has in the layout of these elements. In order to avoid this difficulty, but still provide the ability to create forms in this fashion, we impose the twin Draconian restrictions that (1) forms must be specified as HTML tables, and (2) any elements that appear in the form must exist in a database of allowable form elements. Limiting forms to tabular formats curbs some aesthetic possibilities, but with a certain amount of creativity, this is not such a huge restriction. Restricting the elements that can be used is also not such a great loss, since HTML already severely limits the types of input elements that can appear on forms. The WebSubmit form element database contains most of the core HTML input elements; image maps and embedded code (like Applets) are the only basic elements not currently supported.

The goal of this approach is to make generation of the form straightforward and simpler than constructing the HTML by hand (especially if you don't know HTML or how to build form input elements!). A collection of form element primitives is specified in a *form element database*, which contains atomic and composite elements and also controls how data is validated for these elements (Section 3.2). The atomic elements in this database correspond to the core HTML input elements, along with a variant designation that indicates what type of data the element expects. Composite elements are simply combinations of atomic elements. The form itself is specified in a specialized form database; each record within this database specifies a cell in the table that makes up the form. Construction of the form database requires developers to specify elements as they appear in the form element database (i.e., call the constructor procedures for these elements with the proper arguments). The procedures to construct form elements are calls to routines in the `cgi.tcl` library, with some supplementary code to name elements and control other aspects of their construction.

3.2 Data Validation

Creating the form is the (relatively) easy part. The challenging part is transforming an undifferentiated mass of input data into something meaningful and ultimately useful, and doing this without having to write an extensive body of code. The first step in this process is *data validation*: analyzing input data and attempting to determine whether or not a user has submitted garbage or not. This type of error checking is common in applications with graphical user interfaces, and has been mentioned before in the context of Tcl/Tk.[14] For example, suppose a form element exists that asks for the number of widgets produced by the WebSubmit Widget Company. Clearly, this field should contain a positive integer (assuming success on the part of the company), but the user might misunderstand the purpose of the field or just mistype the input. We trap these simple input errors using a collection of *error-checking primitives*. For each variant of a form element, the set of error checks to be performed on the data in that element is specified in the form element database. In the example above, the form element would be a variant of text entry called number entry, and error checks would verify that the data was a positive integer. An error would be generated if the data didn't pass the validation test(s). More complex error-checking primitives exist, such as a group to perform checks on remote file system data. For example, some form elements might request the names of files on remote systems. As part of the validation process, we include primitives that check to see whether such remote files exist.

3.3 Tasks: Putting the Pieces Together

After data validation, we still have an undifferentiated mass of input data, but at least we know it is a *reasonable* mass of data. Now we have to take this clay and mold it into a task that can accomplish something useful. Tasks can be reduced to a three step process:

1. Process input, applying semantics to the elements that are meaningful for the task at hand

2. Build a job script that executes the required commands

3. Execute the script on the remote system and collect output

Several implementations can be used to accomplish this task. One solution is to create some sort of Tcl API for accomplishing components of tasks, and then write Tcl code to specify how to accomplish the whole task. Another approach would be to specify a simple list of primitive tasks to be performed, each of which is chosen from a fixed set of elements. The overall task is then accomplished by combining these primitives in a database and calling them sequentially. The former approach requires developers to learn the job scripting API and create structured Tcl programs. The latter method is somewhat inflexible, if it is not complicated by adding elements equivalent to control structures (conditionals and loops); it is, however, more amenable to rapid development.

In WebSubmit, task construction is accomplished using the latter approach, with a task database that acts as an algorithm for the task. Records in the task database are executed in the order specified; each separate record specifies a command primitive selected from a group of available commands (contained in a task element database). Task primitives might specify files to be transferred, templates to be processed to create needed input files, or input data manipulations. For example, suppose an application module is used to submit jobs to a batch queuing system (e.g., Load Leveler). The task database would specify the following:

1. Manipulate keyword input as necessary

2. Construct queuing system keywords and shell script

3. Transfer the control scripts to the remote system

4. Execute the shell script to submit the job request

5. Collect output

It is not difficult to imagine arbitrarily complex tasks that might not fit within the framework of existing task primitives. At this point, developers would need to augment the task element database to suit their local needs. This does not meet the ideal of completely automatic processing, but at some point there is a trade-off between complexity/flexibility of the application and ease of use.

4 Tcl and WebSubmit

WebSubmit is composed of several separate bodies of Tcl code: (1) CGI scripts, (2) service procedures, and (3) databases and configuration data. The code assumes Tcl 8.0, since namespaces and some other features of 8.0 are used throughout. CGI scripts are designed around Don Libes' excellent `cgi.tcl` library.[3] The remainder of the code was written by the WebSubmit development team.

4.1 Data Encapsulation: Packages and Namespaces

In an effort to support some level of data hiding and to make loading sets of routines more transparent, service routines are broken up into functionally-distinct Tcl packages. All of the procedures and data for each package are then encapsulated inside a namespace whose name matches that of the package. In addition, each database is stored within a specified, common data structure in its own namespace. In some cases, there is a correspondence between packages and databases (e.g., `auth.db` matches the authentication package `wsAuth`). The current collection of WebSubmit packages and databases is specified in Table 1.

Table 1: WebSubmit packages and databases. A '-' indicates that there is no database corresponding to the specified package.

Package	Database	Description
webSubmit	-	Main WebSubmit package
wsMaster	master.sdb	Package for master database
wsDatabase	-	Generic database routines
wsAuth	auth.db	Authentication package
wsNetwork	-	Procedures for interaction with remote systems
wsUtil	-	Utility routines used by all packages
wsCGI	-	WebSubmit extensions to *cgi.tcl*
wsError	errors.db	Error-checking primitive routines
wsForms	forms.sdb	Procedures for automatic generation of forms
wsFormElements	formElements.db	Primitives for form elements
wsTasks	tasks.db	Procedures for automatic generation of tasks
wsTaskElements	taskElements.db	Primitives for task construction

4.2 Namespace Variable Aliases

One aid that Tcl provides for references to variables that lie in different scopes is the `upvar` command. Namespaces add another layer of complication to Tcl when it comes to scope resolution. Fully-qualified names can be used to access variables in namespaces other than the current or global namespace, and the `variable` and `global` commands can be used to make reference to variables within these latter two. In order to ease variable referencing in different namespaces, we have created a command that performs the same function as `global`, but works for arbitrary namespaces. The command syntax is

alias \$nameSpace [var_1 var_2 ... var_n]

where \$nameSpace is the name of the namespace in which you wish to create a reference, and $\text{var}_1 ... \text{var}_n$ are the variable names you wish to reference by the same name in the current scope. It is roughly equivalent to a procedure that performs the following:

```
foreach var [list var1 var2 ... varn] {
   uplevel "upvar 0 ${nameSpace}::$var $var"
}
```

In addition to performing this association, the procedure verifies the existence of the specified namespace and then creates the specified variable within this namespace if it does not already exist. This eliminates some of the complexities that arise when trying to manage variables in a (possibly large) collection of namespaces.

4.3 Databases and Serialization

As stated previously, the bulk of the data related to WebSubmit is stored in a collection of simple ASCII databases. Read and build routines in WebSubmit read and parse these databases and store them in the appropriate data structures. One namespace per database encapsulates this data, which includes not only the database itself, but supplementary information about its structure. Since WebSubmit is a CGI application, performance is a consideration, and there could be significant overhead incurred by constantly reading databases and building the corresponding data structures. For this reason, a method for database serialization is used to construct a Tcl representation of a database. This is similar in spirit to object serialization in Java and is a technique that has been utilized previously with Tcl.[15, 17]

The current implementation of WebSubmit stores databases as arrays keyed by a combination of the record key and the attribute name. Database attributes are dynamically obtained from the database itself, so there is complete flexibility in terms of the fields of the database. A sample database with one record and three attributes would appear as follows:

```
::DB_ATTRIBUTES:: key:idName gender workerStatus zone
thx1138 : Male : Drone : Sector 6
```

The data for the entry in this database with an `idName` (record key) of `thx1138` would be stored as `DB(thx1138,gender)`, `DB(thx1138,workerStatus)`, `DB(thx1138,zone)`. Since all of the data are stored in a single array, it is quite simple to write this data to a file after it has been read such that `source`-ing this file at a later time would be equivalent to reading and restructuring the database. The structure of this serialized database consists of a collection of Tcl statements that represent the database:

```
# Serialized version of three attribute database with one record
#   Data encapsulated in namespace dbNameSpace
namespace eval dbNameSpace {
   set DB(thx1138,gender) "Male"
   set DB(thx1138,workerStatus) "Drone"
   set DB(thx1138,zone) "Sector 6"
}
```

By comparing timestamps, we can write the serialized database whenever the regular database has been modified. Otherwise, the action of reading and restructuring the database is reduced to a single `source` command.

4.4 Source Code Control

Any project that involves multiple programmers creates some difficulties with code management and testing. Large projects involving script-based languages like Tcl present some slightly different challenges; in order to provide a flexible development system, we have written a Tcl wrapper utility that is used in conjunction with the Source Code Control System (SCCS). SCCS provides a means for controlling source code, but for several reasons, it is not convenient to use with scripting development efforts. There is no separate executable in scripting environments as there is for compiled languages, and developers usually need the entire distribution to debug, even if they are only working on a small section of code. This creates the need for multiple copies of source code spread across a tree of developer directories, one for each developer. An additional "clean" copy of working scripts can be maintained in a separate public directory. The need for multiple directories for source code creates the additional complication of modifying any path-specific information in the source code. Also, in the case of a codebase for CGI scripts, there are file permission issues that need to be addressed.

For WebSubmit, we constructed a Tcl wrapper script that uses the Source Code Control System (SCCS) as its base for script control, and supports public and developer directories for source code scripts. SCCS commands are invoked as if there were no directory hierarchy; the SCCS wrapper script hides all of the complications associated with multiple development directories. All SCCS files are kept in a common root directory; the wrapper script is responsible for moving files between this root directory and the appropriate location (public or developer directory). After moving files as needed, file paths are modified to suit the new location of the source code and file permissions are updated. Public distributions can be created when all source code is checked in. Either full or partial updates of the public distribution are supported.

5 Future Directions

Certain aspects of WebSubmit are still works in progress, and implementation details could be subject to change during the completion of the development cycle. The future of WebSubmit will, in part, be driven by user experiences with the software. Requests for additional modules and services will definitely be supported, as will bug fixes. Based on user input, we should also be able to more accurately assess any CGI performance issues and optimize the code accordingly. Object-oriented methodologies (such as those used with `[incr Tcl]` [18]) are being investigated intensively, since the function of WebSubmit lends itself to using object-oriented paradigms. The rapid nature of development of the Tcl core presents problems for the developers of language extensions (like `[incr Tcl]`), hence we have chosen to consider adopting these technologies when there is more stability in the Tcl development world. There is also a possibility that certain aspects of WebSubmit will be ported to a Java environment. This may be unnecessary if the use of Tcl browser plug-ins becomes more widespread, or if `[incr Tcl]` becomes a part of the Tcl core.

6 Acknowledgements

The authors would like to acknowledge Robert Lipman and Katherine Pagoaga for their previous work on the project. We would also like to acknowledge Don Libes for many useful discussions on `cgi.tcl`. James Dray of the NIST security division provided useful insights into the WebSubmit security architecture.

References

[1] Robert R. Lipman, Judith E. Devaney, "WebSubmit - Running Supercomputer Applications via the Web", *Proceedings of SuperComputing 96*, November 1996, Pittsburgh, PA (`http://www.supercomp.org/sc96/SC96PROC/POSTER.HTM`)

[2] John. E. Koontz, Ryan P. McCormack, and Judith E. Devaney, "WebSubmit - A Paradigm for Platfom Independent Computing", presented at the *Workshop on Seamless Computing*, Reading, England, Sept. 1997 (`http://www.ecmwf.int/html/seamless/wkshop.html`)

[3] Don Libes, "Writing CGI Scripts in Tcl", *Proceedings of the Fourth Annual Tcl/Tk Workshop '96*, Monterey, CA, July 10-13, 1996, USENIX Association (Berkeley, CA)

[4] R. McCormack, J. Koontz, and J. Devaney, "Seamless Computing with WebSubmit", *Concurrency: Practice, and Experience* (submitted)

[5] Jim Almond, "Seamless Computing via the World Wide Web", Unpublished white paper (`http://www.ecmwf.int/html/seamless/white.paper.ps`)

[6] W. A. Wulf, C. Wang, and D. Kienzle, "A New Model of Security for Distributed Systems", University of Virginia CS Technical Report (`http://www.cs.virginia.edu/~legion/papers/CS-95-34.ps`)

[7] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", *Int'l Journal of Supercomp. Appl. and High Perf. Computing*, 11(2), 115-128 (1997)

[8] Nathan Neulinger, `cgiwrap` home page (`http://www.umr.edu/~cgiwrap`)

[9] Bruce Schneier, *Applied Cryptography*, 2nd Edition, John Wiley & Sons (New York, 1996), pp. 185-187

[10] Netscape SSL Overview (`http://home.netscape.com/newsref/std/SSL.html`)

[11] Scott Leerssen, `certutil.c` (`personal.bellsouth.net/~leerssen`)

[12] Tatu Ylönen, SSH Web site (`http://www.cs.hut.fi/ssh`)

[13] Internet Draft for SSH (`http://www.ietf.org/ids.by.wg/secsh.html`)

[14] S.D. Mullerworth, "A Flexible GUI Design System", *Proceedings of the Fifth Annual Tcl/Tk Workshop*, June 14-17, 1997, USENIX Asociation (Berkeley, CA), p. 163

[15] De Clarke, "Dashboard: A Knowledge-Based Real-Time Control Panel", *Proceedings of the Fifth Annual Tcl/Tk Workshop*, June 14-17, 1997, USENIX Asociation (Berkeley, CA), pp. 9-18

[16] Gaussian94 Home Page (`http://www.gaussian.com`)

[17] Michael McLennan, unpublished course notes for "Building Applications with Tcl/Tk", AT & T Design Automation, AT & T Bell Laboratories (1995)

[18] Michael McLennan, "Object-Oriented Programming with [incr Tcl]", in *Tcl/Tk Tools*, O'Reilly & Associates (Sebastopol, 1997), p. 7