Proceedings of the 7th USENIX Tcl/Tk Conference

Austin, Texas, USA, February 14–18, 2000

TCL/TK: A STRONG BASIS FOR COMPLEX LOAD TESTING SYSTEMS

Ahmet Can Keskin, Till Immanuel Patzschke,
and Ernst von Voigt

Tcl/Tk: A Strong Basis for Complex Load Testing Systems

Ahmet Can Keskin, Till Immanuel Patzschke, Ernst von Voigt

Patzschke + Rasp Software AG

Abstract

This paper describes a Tcl/Tk-based load testing environment which was developed for Deutsche Telekom, Europe's largest telco carrier and online service provider. Deutsche Telekom uses the system to ensure a high quality of service and availability.

The paper explains the complex requirements of load testing and gives a detailed overview for the extensive use of Tcl/Tk within the system.

1 Introduction

The Center for Internet and Data Network Platforms (ZID), a division of Deutsche Telekom, provides the network infrastructure over which the largest Online Service in Germany, T-Online, is operated. Deutsche Telekom is Europe's largest and the world's third largest telecommunications carrier, and its online service has more that 3.3 million customers - which grew at a rate of 42 per cent during the last fiscal year.

ZID is responsible for ensuring that all hardware and software components of an access network function coherently and can withstand the heavy demands of hundreds of thousands of simultaneous users, both prior to deployment and during operation. ZID must also be able to deliver a guaranteed performance, bandwidth, and availability.

ZID has used the load testing technology for many years to ensure that the online services it provides maintain the Quality of Service (QoS) level that Deutsche Telekom's customers have come to expect. The objectives for those early test systems were functional testing, throughput and general performance testing for a BTX-based network. Recently, enhancements for the testing of the latest internet technologies, such as ISDN or ADSL, have been developed. Critical test factors were defect removal before deployment, and a guaranteed highly responsive system.

Since each new development stage has been accompanied by load testing, Deutsche Telekom was able to meet these goals and deliver a top quality service to its customers. The load testing technology used has evolved along with the network technology it is required to test.

Load testing is the general term used to describe placing a network infrastructure under stress to test its behavior in a production environment. We developed a fully scriptable load testing system based on Tcl/Tk. The basic principle in the design of this system is the Automatic User, abbreviated to AT - the short form is derived from the German term "Automatischer Teilnehmer" [1].

An AT simulates the actions of a human user (e.g. hitting web pages). Every AT can generate a certain load on the network, dependent on the connection (Modem, ISDN, ADSL). The load varies from 28.8 Kbit/s (Modem) over 64 Kbit/s (ISDN) to 768 Kbit/s (ADSL). The latest release of the AT system allows a maximum of 225 ATs to be run simultaneously, which are then managed and controlled with a graphical user interface, providing a single point of control and monitoring. The number of ATs used in a testing environment is virtually unlimited and only depends on the hardware configuration.

Both the control interface and the AT are implemented in Tcl/Tk. The scripting language Tcl proved to be an ideal platform for implementing the AT, which freed us from designing a new high level language for the specification of test scenarios and implementing a runtime environment to drive such scenarios. Tcl's mechanism of slave interpreters was sufficient to achieve these goals. Tcl/Tk also supported rapid development of the graphical control interface.

2 Background: Load Testing

To support its online services nationwide, Deutsche Telekom operates more than 180 POPs all over Ger-

many. In each of the POPs, networks with specific hardware and software systems have been installed, which ensure the user's access to the internet. The introduction of new online services like ADSL requires the development of a new infrastructure for the access network, which has to be installed nationwide in all of the POPs. The requirements for such an access network are:

- High bandwidth
- Low costs
- High reliability

Designing such a network can be a challenging task. For providers of online services like Deutsche Telekom with 120 million internet connections per month, it is important to detect and eliminate deficiencies and performance bottlenecks before they deploy an access network nationwide. This is especially important because additional improvements can be very expensive, since all the POPs have to be upgraded and the users might not be able to use the services.

Deutsche Telekom has established two operation centers in Darmstadt and Ulm in order to test and optimize the entire system in a production-like environment prior to deployment. These centers are fully equipped with web servers, connection hardware, and all equipment for the planned network installation in a regional node (POP). Load testing is then performed using the AT on the complete network. Network components, including software, which allow access to the internet platform, are tested for their load-bearing capacity and stability.

Test results are used to make decisions on how best to optimize performance and deliver new functionality. Decisions such as replacing a router with a larger one are made. The test results are also used to verify whether or not the vendors components meet up to their claimed capabilities. The vendors providing equipment for the network often use the results produced by the AT (in conjunction with their own tools) to isolate defects, verify changes and fixes.

Once the developers at ZID are satisfied with the resulting network, it is deployed in more than 180 regional nodes throughout Germany. In addition the AT is also used in these locations to ensure their individual QoS once operational on a specific hardware basis. ZID relies heavily on the AT analysis tools to compare different load scenarios at different times of day. One of the major benefits of the AT is that all testing and analysis work is concentrated in one location and controlled from a single computer.

3 Design & Implementation

3.1 Requirements

In the past years, the ZID has been performing load tests with various tools for developing and optimizing their access networks. Experience has shown that load tests are most effective when the real conditions are simulated as closely as possible.

Let us consider a typical user, who wants to connect from his/her PC to the internet: He or she might be a subscriber to an Internet Service Provider performing transactions such as surfing the web, doing FTP file transfers, or sending e-mail via an SMTP client. Another typical user might be using an e-commerce site to make purchases or making queries to a web-enabled database application. Therefore, her or his essential activities are:

- Establishing a connection: The user dials his ISP's telephone number, connects, and logs into his account with a password.
- Doing online transactions.
- Closing the connection: The user logs off and closes the connection to his ISP.

To test the capacity of an access network (e.g. number of concurrent users), the load testing system had to perform realistic emulations for concurrent user activities, like HTTP, FTP, etc. Further requirements included:

- Single point of control (GUI)
- Measurement of performance criteria such as response time and throughput
- Flexible, yet easy-to-learn scripting language for defining test scenarios
- Scalability regarding the number of simulated users (AT)
- Extensibility, e.g. easy integration of additional network protocols (SNMP, RTSP, etc.)
- Online monitoring
- Distinct physical connection for each simulated user using different protocols (e.g. PPPoE)

3.2 Design Decisions

To meet the above requirements, we opted for an open platform providing support for almost all available hardware: Linux. In terms of a flexible scripting language, we looked at Perl, Python, and Tcl. Although Perl is pretty popular in the "system administration community" it isn't as easy-to-learn and handy as the other languages (especially for end-users.) A second important point was the GUI component - an area where Tcl/Tk is

a natural choice, since both other languages lack native (i.e. out-of-the-box) GUI support and use for example Tk instead of a “native” solution.

Since the idea of the AT implies managing hundreds of different processes - including inter-process communication between them - easy-to-use and powerful network support was another important criterion. Last but not least the “glue” argument was very important. Tcl is the perfect tool for tieing different components together without creating monstrous extension libraries.

The AT’s architecture is based on software agents that simulate a human user and are controlled centrally over a graphical user interface. (For details please refer to section 3.3: Architecture and System Components.)

Due to the flexibility of the Tcl/Tk environment, most of the entire system’s components have been implemented using it:

- Rapid prototyping: When starting the development, we were unable to predict future technological advances and we did not know exactly which features the simulated user should support. It was thus important to have fully functional software agents whose features and functionality could be improved step by step.
- Description of scenarios: Tcl makes it easy to specify the activities of a test scenario because it is a high-level language providing all necessary control structures (if, while, etc.) to simulate every kind of sequence of a user’s actions. This freed us from designing another scripting language for test scenarios and implementing a runtime environment.
- Visual Control: The management of the simulated user had to be possible from one central point with a graphical user interface - an easy job for Tk.
- Protocol modules: The simulated user should perform transactions based on the FTP and HTTP protocols. We were able to use existing Tcl packages that served as a basis to implement the desired functionality. Furthermore, Tcl’s encapsulation of basic system functionality (like TCP/IP sockets) facilitated and simplified protocol implementation.
- Automatic User (AT): The simulation of a human user could be handled by a software agent which ran on a session host with a physical connection to the access network, and which was controllable from a central point (control host). Tcl’s mechanisms for inter-process communication via sockets makes it easy to realize such distributed applications.

3.3 Architecture and System Components

Figure 1 shows an architectural overview of the load testing system. Each block in the diagram represents a logical software component.

A **load scenario** describes a complete set of transactions carried out by a specific number of users connected in a particular way to a network over a given period of time. Load testing involves managing a potentially large number of ATs executing load scenarios. It is especially important to have a means to control all aspects of testing from a central location.

The **AT controller** component allows the creation of an interactive environment for defining, driving, and coordinating a load test scenario. The AT controller is designed using a set of building blocks which allow a complete definition and management of a scenario. These blocks include:

- access to a repository of Session Scripts
- a means to define connections to the tested system
- configuration of the sessions
- session management during execution of the scenario

The **script repository** contains all the scripts that have been defined by the test designer. A given script can be assigned to one or more automated users for a given scenario. The AT provides example scripts which may be extended and/or parameterized to create unique automated users. Also the AT provides libraries implementing basic behavior patterns of real online users to reach a higher level of abstraction. A high level of parameterization is possible by allowing the script, to reflect the actual behavior of a typical user without having to create a new script even when the basic actions taken are not the same. For instance, a time-out can be specified to simulate a user cancelling a web page fetch after 5 or 10 seconds. Pauses (or sleeps) can also be defined. The specific URLs fetched or the username/password required as input for certain operations can all be parameterized. The parameterization is done by the test designer either programmatically to the AT controller or via a GUI.

An **automated user** simulates a real user performing typical operations. A typical user might be a subscriber to an Internet Service Provider carrying out transactions such as surfing the web, doing HTTP GET or POST operations, downloading files using FTP, or sending e-mail via an SMTP client. Another typical user might be using an e-commerce site to make purchases or make queries to a database application through a web front-

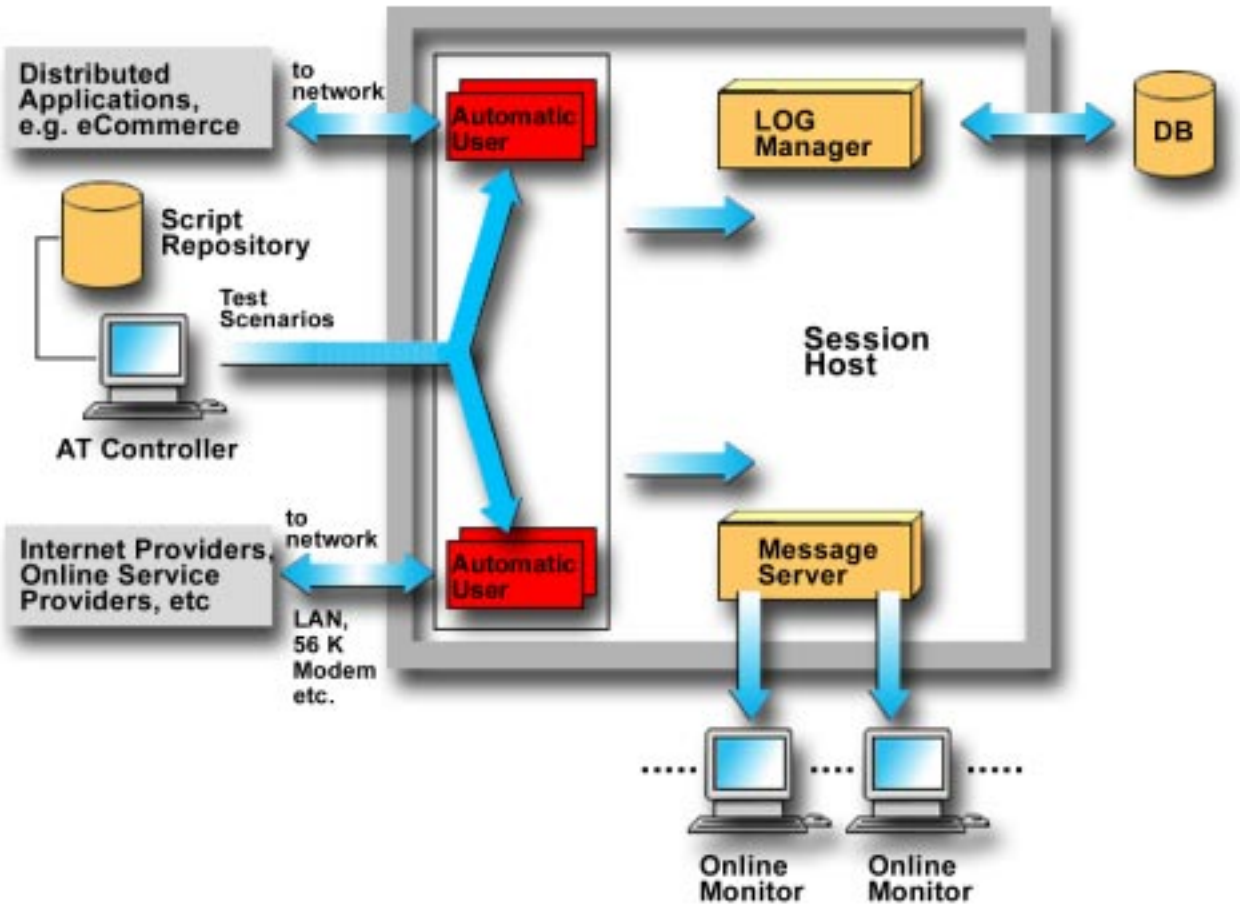

Figure 1: Architecture of the Automatic User

end. The AT creates a run-time environment in which a session script is run performing a work session. The AT builds the connection to the network, carries out the operations specified in the session script and disconnects from the network at the end of the session. Since the AT can be configured to connect via an analog or ISDN modem, a LAN or WAN, amongst others, the throughput measured is what a real user could expect to see. The AT is also responsible for forwarding test results to the message server and the log manager for monitoring and analysis. It also provides status information to the Session Control upon request, including such things as whether a session is currently executing, whether a script is loaded, etc.

The **log manager** receives messages from the various ATs running on a session host [2]. It adds a time-stamp to each entry and, depending on any filtering of messages that has been set, writes the message to the designated logging facility. Messages may be filtered based on the message type. Current types of messages are: trace, log, debug, performance, error.

Like the log manager, the message server receives messages from the various ATs running on a session host. It filters test results and passes them to the connected online monitors where the tester can review the progress of the load scenario and take actions based on the results. Filtering is specified by the online monitors. For instance, an online monitor may specify that it is only to receive trace messages only.

The **online monitor** makes it possible to take action on tests as they take place. For instance, it may be necessary to create additional load over a specific connection to stress the system in a new way based on the results being produced by the currently running sessions. Multiple online monitors may be connected to a session host which allow customized viewing of results. The number of monitors and the purpose of each is specified by the test designer. The monitors connect to the message server running on the session host. Each online monitor can filter the different types of messages processing only those of interest to it. A monitor specifies to the message server what types of messages it wishes to receive.

Several online monitors can be created, each tailored to the specific needs of a test installation. One example is a specialized online monitor listening to trace messages only, sending an e-mail and/or SMS message when a URL specified is not reachable.

Another likely configuration would use an online monitor that views only debug messages which need to be handled by the creator of the load scenario and another online monitor that views only performance messages used by the network architect to identify bottlenecks and refine the network configuration.

All online monitors can be connected to a graphic interface providing dynamic graphic views on result data. The entire dynamic interface is built using the Tcl/Tk based product GIPSY [4], allowing easy customizability and creation of dynamic data visualization.

3.4 Distributed Components and Communication

Since Tcl has easy-to-use communication commands, building a distributed client/server system is simple. On top of standard Tcl we implemented a communications layer as basis for RPC and simple data streaming (for fast transmission of status data.) Additionally an "Application Name Server" (ANS) provides host/port lookup services.

Tcl-DP [3] appeared too "heavy-weight" for the above purpose, so we decided to use "vanilla" Tcl to realize the entire communication layer.

Needless to say that all components of the AT use the communications layer, allowing easy expansion and distribution of functionality. System Configuration

3.5 System Configuration

A typical installation of the load testing system consists of one control host and a number of session hosts. The control and session hosts are connected through an ethernet LAN, which we call AT backbone. Several ATs run on each session host. The number of ATs per host depends on the connection to the ISP or the Internetwork (ISDN, Ethernet, ADSL, Modem etc.).

The current configuration at ZID uses 4 4-port ethernet network adapters per PC emulating ADSL lines to an Access Concentrator (AC), providing 15 session and one backbone port. The backbone connects 15 PC running ATs to one AT controller, acting as single -point-of-control for the entire system.

4 Automatic User

As stated above, the AT simulates a real user performing online actions like HTTP get or FTP file transfers. Implemented as a software agent, the AT is built upon four major modules: communication, connection, logging and script engine.

The communication module encapsulates all communication aspects like a RPC server socket, connecting to the controller or the message server and sending them status and log messages. The AT receives and executes commands from the AT control via the RPC server socket.

The connection module provides all functions to connect to a network and authenticates a user with his name and password. The logging module implements functions to log all actions of an AT to a plain file, a database, send log messages to syslog daemon or the log server. Finally the script engine provides a runtime environment to drive session scripts.

4.1 Session Script

In order to simplify the development of the session scripts and to reach a higher level of abstraction, Tcl libraries are provided. They already implement a basic behavior. This is illustrated in the example below. Listing 1 shows a sample session script. It is a simple example simulating a user who establishes a connection to the ISP via PPPoE, gets some URLs and disconnects.

```
# Sample Session Script
#
set ::url1 "http://192.9.220.240/test/600k.gif"
set ::url2 "ftp://193.156.56.10/2400k"

# set basic connection parameters
access::config -trymax 3 -trydelay 300

# define bandwidth paramters, i.e. if
# bandwidth<30 and times>=4 ...
access::perf::config -threslow 30 -maxnumlow 4

# session main
proc run {} {
    # connect using PPPoE
    access::connect -p pppoe -user joe \
    -passwd pizza
    # get 600k.gif via HTTP
    access::perf::get $::url1
    # get 2400k via FTP
    access::perf::get $::url2
    # end session - disconnect
    access::disconnect
}
```

Listing 1: A sample session script

If an action (e.g. `access::connect`) fails, it is repeated several times (`::try(max)`.) If it still fails, an alert is sent to the online monitors and the log manager. Then the action is repeated after a defined number of seconds (here `::try(delay)`) seconds) until it is successful.

Additionally, a threshold value for the data transfer rate can be defined (`::thres(perf)`.) If the transfer rate drops below the threshold value more often then a defined limit (`::thres(num)`), an alert is raised.

4.2 Script Engine

The script engine is a Tcl module providing a runtime environment for session scripts. To drive a session script the script engine first reads the Tcl source of the session script from the script repository. Then a slave interpreter is created and initialized. In the initialization phase local variables are declared in the slave interpreter, aliases to commands of the master interpreter are created, protocol extensions loaded. Finally the session script is evaluated within the slave interpreter.

To start a session, the script engine invokes the `run` procedure of the session script. Stopping a session means deleting the slave interpreter ignoring the resulting error messages.

Session scripts are written in Tcl using behaviour libraries. Each script has a `run` procedure. The `run` procedure may be executed multiple times to simulate multiple consecutive sessions. All statements outside of the run procedure are executed once per session.

The script writer may define trace messages that will be sent to the message server and log server. Within the script any combination of transactions using the supported protocols can be performed as well as all standard Tcl operations.

It is also possible to set variables in the script prior to running it. This allows customization of a single script logic for different users who perform the same operations but on different targets including: number of run repetitions, transaction time-out intervals, and level of tracing.

For FTP and HTTP we use extension implemented in Tcl. The http extension is built on top of the standard HTTP package coming with Tcl. Ftp is built on the FTP package of Steffan Traeger [2]. The FTP and HTTP packages had to be modified in order to measure the response time and throughput and to provide distinct time-out mechanisms for the connect and for the data transfer phase.

5 AT Controller

The management and control of both the ATs and sessions are accomplished via a graphical user interface (figure 2) which gives the user an overview over the ATs states. The user may start any test scenario and monitor the load tests and their results.

The controller consists of an LED area that represents the ATs, a command input field, a history list, and a status window.

The LED area displays three states for every AT: script status, connect state and a “runs left” counter. The script LED denotes whether a script has been loaded, started or stopped. The connect state indicates one of the following: if the AT is connecting to a network, if it is already connected, if it is about to disconnect or if it is already disconnected.

Controlling the AT means choosing from two options: Direct input using the command input field or selecting the ATs with the mouse cursor and clicking the command button.

The LED area is sensitive and the user may choose the AT to which a command is sent using the mouse. Additionally, he may re-activate commands from the history list. The execution of a command is shown in the status window. The controller is connected to the ATs and may send any command to them via RPC.

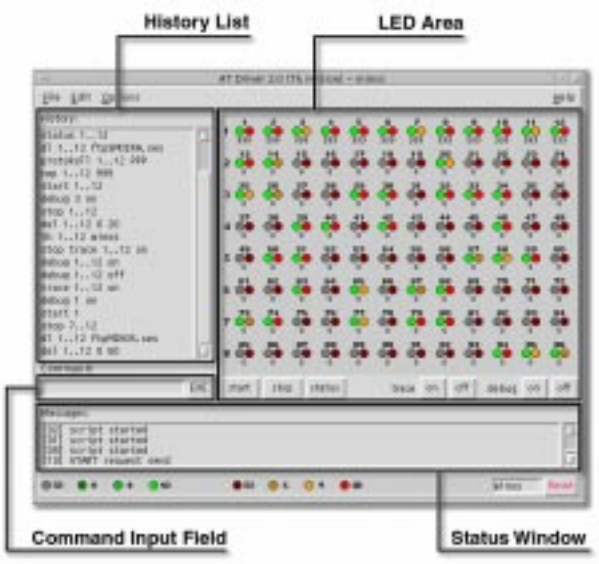

Figure 2: AT’s user interface. For description, please see text above.

6 Experiences with Tcl/Tk

This paper described a fully scriptable load testing application for networks. All crucial components are implemented with Tcl/Tk. The use of Tcl/Tk allowed a development process that is best characterized with “development by prototyping”, and thus enabled us to present a working version to the customer very early in the project. As the project progressed, we could build fully working prototypes and test them under “real world conditions”. This helped us to better understand the requirements and - even more important - to find and eliminate errors in the design very early. Last, but not least, we could implement changes which our customer requested after the design phase with an effort bearable for both sides.

Another important aspect of the “development by prototyping” was that the testing engineers were able to improve the design of the access network based on the test results of prototypes, even in an early development phase. A good example is the ADSL modem-access implementation history. Any access method involving substantial user configuration of their modem or personal computer can hinder rapid, widespread consumer adoption of high-speed services such as ADSL. The first approach, which was provided by a well-known vendor of network components, was a combination of a DHCP client and a HTTP-Browser. The user was required to configure DHCP on his Windows client and open a URL with his Browser to get a login page. After typing user name and password the connection to the internet was established.

As the test engineers began load tests, they detected instabilities of the access network components. The implementation was neither stable nor would it allow more than 28 concurrent logons. These results were reproducible so ZID could reject this approach and force its network solution provider to develop an alternative solution.

The next and up to now final solution for client access method was point to point over ethernet (PPPoE). Because PPPoE had been a rather new protocol there were not clients available at that time, especially not for the Linux operating system. So, we had to implement a client on our own and integrate it into the load testing system. This kind of change was only possible due to Tcl’s excellent “gluing” capabilities (and the flexible architecture, of course), allowing ourselves to keep most of the existing system without any changes for the new protocol.

During later prototyping phases the system was able to detect problems concerning the stability and the bandwidth of the whole access network solution. ZID’s hardware vendor and solution provider had to admit that neither the ATM switching component nor the PPPoE server performed within specification (50Mbit/s vs. 150Mbit/s, 56 vs. 90 sessions.)

Another important aspect for our customer was that we did not modify the Tcl kernel. All of the requirements have been solved with standard Tcl/Tk. C was only used in performance critical areas like the PPPoE protocol handling (i.e. handling Ethernet packets.) Using standards - like Tcl/Tk - without modifications proved to be an important issue, especially regarding the system’s acceptance.

It proved that Tcl/Tk was and is the ideal solution for the implementation of the AT, since we were able to meet the customer’s requirements effectively and react in a timely manner to design changes during the project. Tcl offered an easy-to-learn and flexible language that allowed us to provide a full featured language gaining more and more enthusiasm at ZID (all of the test engineers are non-programmers.) All that was possible without developing a language ourselves by simply using what was already there.

Tcl/Tk helped us to develop a commercial product in a short time and - even more important - to release it to our customer right in time. From our customer’s point of view, Tcl/Tk helps in achieving his mission critical goals with minimal effort.

References

[1] AT and sm@rtTest

The Automatic User described in this paper is marketed by interNetwork AG.

More information may be obtained from their web site under http://www.internetwork-ag.de

[2] FTP Library Package

The FTP Library Package ftp_lib provides the client side of the File Transfer Protocol (FTP). The package extends Tcl/Tk with commands to support the file transfer protocol like OPEN, CLOSE, LIST, PUT, GET, REGET etc. It's used either to add FTP ability to existing Tcl/Tk applications or to create small FTP scripts that perform tasks without user interaction. It allows automatic up/download processes even up to the mirroring of complete FTP sites.

Information on the web:
http://home.t-online.de/home/Steffen.Traeger/tindexe.htm

[3] Tcl-DP

Mike Perham, Brian C. Smith, Tibor Jánosi. "*Redesigning Tcl-TP*" in Proceedings of the fifth Tcl/Tk Workshop. July 1997.

[4] GIPSY

GIPSY is a Tcl/tk-based platform-independent visualization software which is also provided from PRS.

Information on the web:
http://www.prs.de/int/products/gipsy

Address of the Authors:

Patzschke + Rasp Software AG
Bierstadter Straße 7
D-65189 Wiesbaden

Germany

www.prs.de - info@prs.de