The following paper was originally published in the
Proceedings of the Fourth USENIX Tcl/Tk Workshop
Monterey, CA, July 10-13, 1996

# The NR Newsreader

Jonathan L. Herlocker
University of Minnesota
Minneapolis, MN 55455

# The NR Newsreader

Jonathan L. Herlocker
*Department of Computer Science*
*University of Minnesota*
*Minneapolis, MN 55455*

*herlocke@cs.umn.edu*

## Abstract

NR is a point and click GUI interface for browsing Usenet news. The NR interface is built using the Tk interface toolkit, and coded entirely in the Tcl scripting language. NR was designed as a framework for pursuing research into electronic information browsing. As a result, NR had strong requirements for configurability, extensibility, portability, and performance. This paper describes how those requirements were met through the use of features and extensions of the Tcl/Tk scripting language. The paper also describes some of the information filtering technologies implemented in NR.

## Background

Usenet news is a medium that provides for collaboration and sharing of knowledge on a global scale through a bulletin-board model. Usenet is organized into thousands of newsgroups, each of which represents a general topic of interest. Every message posted to a newsgroup is seen by all readers who subscribe to that newsgroup. As Usenet news has increased its readership to millions, the quantity of text posted daily to popular newsgroups exceeds the amount of information that a person has time to process. As a result, reading Usenet news can be a frustrating process. A user must scan hundreds of subject lines each week in order to locate interesting information.

In an attempt to lessen the workload of users reading electronic news, I am experimenting with agent technologies that help a user locate information of interest. One of these technologies is content-based: an agent that observes and remembers the content of user-selected articles and then attempts to suggest new articles that contain similar content. The other technology, part of the GroupLens[1][7] project, is collaborative based. GroupLens helps a user to locate new information based on the opinions of other users with similar interest. Both of these technologies are described in more detail at the end of the paper.

After designing the architecture of my agents, I found that there were no existing newsreaders that met my requirements for a prototype system. I wanted a GUI-based newsreader to provide more flexibility in creating interfaces for visual feedback. Existing GUI-based newsreaders such as XRN[6] did not lend themselves to easy extension of the graphical interface.

I chose to implement a prototype newsreader in Tcl/Tk, primarily because I knew from previous experience that I would be able to quickly build a working interface. In addition, the Tk text widget had the desirable property that it both understood how to lay out text efficiently and supported embedded graphics and controls.

After working part time for three months, I had a working prototype that supported my agent assistant. However, I was unable to do a suitable experiment to determine the usefulness of my agent. Users were unwilling to spend their time using a newsreader that did not support the features that they were used to. Therefore I decided to refine my prototype into a full-featured newsreading application that not only supported the most popular newsreading features, but was highly extensible, allowing it to be a testbed for agents such as the ones I designed. Having a liking for simplistic terms, I called the newsreader *NR*. Figure 1 shows a screen shot of NR browsing the newsgroup comp.lang.tcl.

## Requirements

There were four major requirements that would make NR possible and successful.

1. Ease of coding
2. Configurability
3. Good performance
4. Portability

### Ease of Coding

NR was a not a project that I could afford to spend all my time on, so developing it required that it take minimal effort of coding. Using Tcl helped to satisfy that requirement. Tcl shortened the development time considerably over that of a compiled language such as C.

Tcl is an ideal language for a Usenet client, because both the data and communication protocols in Usenet are entirely textual, and require little support for binary datatypes.

One of my constant paranoias is that I will reinvent the wheel every time I write a piece of code. However, I was much more at ease when writing NR, due to the enormous amounts of code that I reused in developing my application. Of the 16,360 lines of Tcl code currently in the NR application, only 6,350 lines were written by me.

The majority of the code I reused came from the exmh mail reader[10]. Organizing code in fashion similar to exmh (as described in [10]) is a good way to create modularization for a language that provides little built-in support for it.

However, one of the frustrations that I experienced when writing NR was that there was no good place that I could go to locate code modules for code reuse. Current Tcl/Tk archives provide a centralized location for Tcl/Tk source code and applications, but these archives focus on applications and not on reusable code modules. In addition, all the sites provide the applications and code modules in a form of a monolithic list with nothing more than a one or two line abstract. A much more structured interface is needed for a Tcl/Tk code modules archive. This ideal archive for Tcl/Tk code modules will be hierarchically organized, so that if a programmer wants to find a common interface element (such as an extended listbox, file selection box, or extended canvas widget), he can move directly to a point in the hierarchy and only browse through applicable modules.The abstract text of all code modules and applications should be searchable, so that users can locate applications of interest, and programmers can locate Tcl code modules if they are unsure of its location in the hierarchy.

The high-level language features of Tcl made the coding and debugging process easy. Memory management bugs, which are my primary source of errors in other applications were non-existent in Tcl. Tcl's use of a single string data-type freed me from much of the work of designing data-structures and made passing data to other

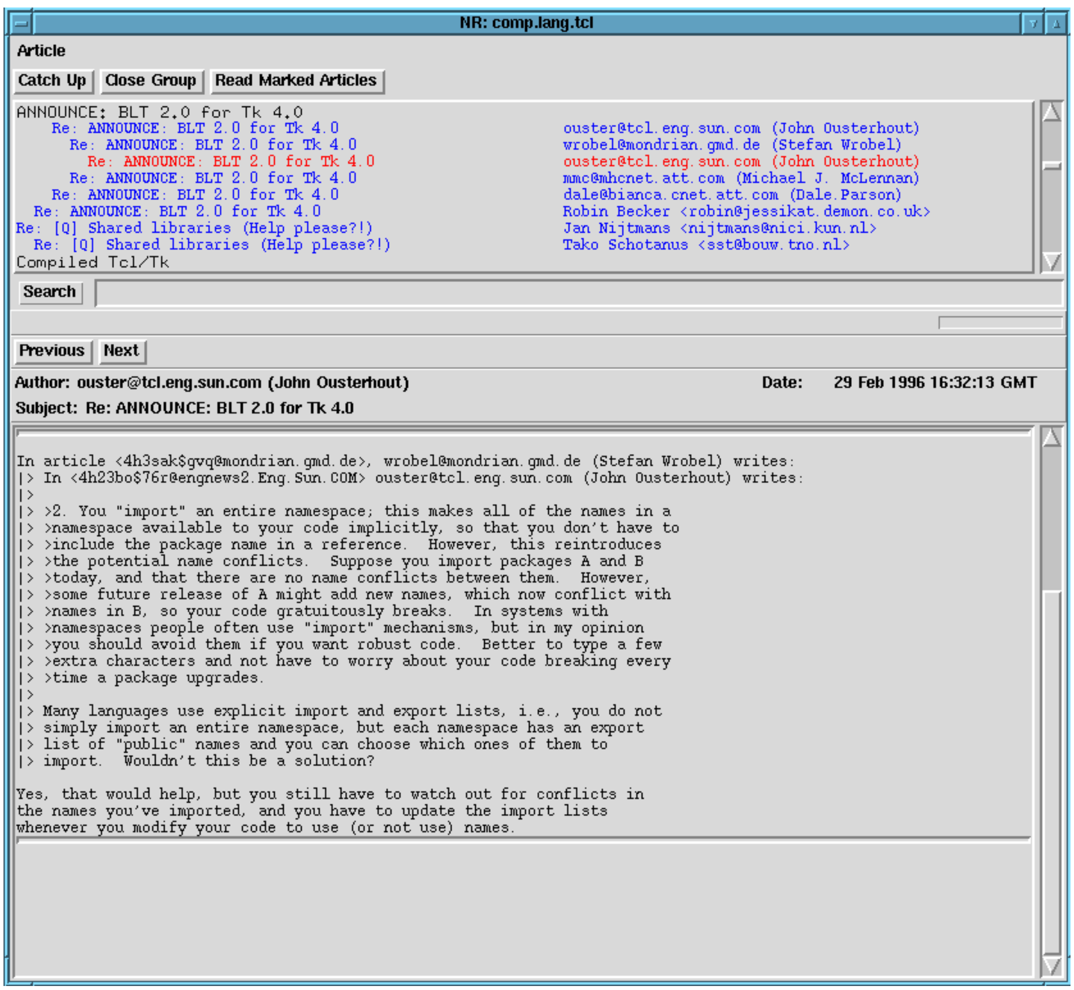


Figure 1: Snapshot of the NR group view window while reading the newsgroup comp.lang.tcl.

people's code modules simple.

Tcl-Prop[2] was a Tcl extension that I found simplified a good portion of the interface code. Some of the most monotonous busy work in programming modeless interfaces such as Tk is the enabling and disabling of controls based on their applicability to the current state of the application. The standard way to handle these controls is to find every location in your code where the state of the program changes and insert a call to the enabling/disabling routine. However, the state can change in many locations in your code, possibly even in somebody's else's code. Another method is to use Tcl-Prop, a data-propagation formula manager. Tcl-prop is a Tcl-only extension which provides a runtime constraint system in some ways similar to that found in Garnet[5]. Tcl-Prop will automatically change the state of the controls whenever the state of the application changes. Setup of the constraint system is simple. You call one procedure to define how the state is encoded, then provide a Tcl script to be executed whenever the state changes. State can be as simple as a variable, or something more complex like window visibility. Tcl-Prop allows you to trigger Tcl scripts on events that occur in code modules without editing the code of those code modules.

**Configurability**

Every user of Usenet news has developed their own habits of browsing electronic news, and many have no desire to change their habits. This was my stumbling block when I sought to obtain useful data to document my agent assistant. So NR had to support commonly desired features and be easily extended to support other features that users might desire. I also wanted to design NR such that it could be easily extended for research involving information browsing such as user trace collection or development of agent assistants. Overall, NR had to be an application that was highly configurable in terms of features and interface.

I chose to incorporate much of the configurability code from the exmh mail reader. The configurability module in exmh provides for customization through the use of options and X-defaults. Users can not only set application variables, but specify new button and menu widgets to be added to the interface through the use of X-defaults. The exmh preferences module also takes a list of available preference options and builds a GUI interface for manipulation of those options. A personal Tcl library allows users to override parts of the exmh code with their own personalized code.

However the model of editing X-defaults and adding Tcl code can be too complex for many casual users, especially if they are not literate in Tcl. These casual users may often wish to take advantage of new features that have been added by others or configurations created by

```
set moduleName "killFromAOL"
set moduleVersion "1.0"
set moduleDescription "This module will cause NR to discard all posts
      from authors at AOL (*@aol.com)"
set moduleHelp "By default,the killFromAOL module will alert the user
      before discarding any posts, but killFromAOL can be configured to
      quietly and indiscriminantly kill all posts from AOL. See the
      killFromAOL preferences for more."
Preferences_Add "Kill AOL" \
      "These options control preferences for the Kill AOL module" {
            killAOL(showStats) \
            killAOLShowStats \
            ON \
            { Show statistics } \
            "This setting controls whether or not a dialog box will
            inform whenever AOL articles are killed, how many articles
            were killed and provide you with the option of overriding."
      }
}
NR_AddKillPattern author {*@aol.com}
```

Figure 2: Example feature module that will cause NR to discard all articles originating from authors at aol.com.

others. To serve their end, NR supports "One-file feature modules." A user who extends the NR interface can share that extension with other users by simply mailing them a single Tcl file. A simple GUI interface allows novice users to install new feature modules easily, as well as inventory, enable, disable, and remove feature modules. In addition, NR supports versioning of modules, to prevent conflicts among multiple versions of the same module.

For example, consider a feature module that causes your newsreader to automatically discard all postings from authors at a specific site. The feature module might look something like the one shown in Figure 2.

A feature module contains four parts: the name and version of the module, a description of the module and help text on how to use the module, a list of user-configurable preferences for the module, and the code to implement the module. A description of the preferences code can be found in [10].

Adding new features to NR is made easy by providing a documented set of hook functions, which allow users to specify Tcl scripts to be executed at key points of the application. Statistics on newsreading can be collected through hooks that are called whenever an article or group is read. Extensions can control how and what articles are displayed through hook functions. Together these hooks can provide information and control to intelligent agent assistants.

**Good Performance**

In order to create a viable user interface to Usenet news, NR has to provide comparable performance to other existing newsreaders which may be written in compiled languages such as C. Speed was a considerable problem with the initial implementation of NR. There were three sources for poor performance: the network protocol, content parsing of text, and insertion of text into the text widget.

Since electronic news is a data intensive medium, the network and NNTP[3] news server were the primary bottleneck. NR attempts to lessen the impact of the delay by using incremental loading. NR incrementally loads newsgroups upon start-up, allowing the user to start reading the first newsgroup as soon as it becomes available, while continuing to download other newsgroups in the background.

Due to the inherent performance limitations of Tcl, post-processing of article text proved that it could be time consuming. In several cases, NR needs to parse large text articles, looking for regular expressions or patterns. Currently there is no good solution for these situations, however in the future, NR may prefetch and preprocess article text off-line. Prefetching will also hide the network delay.

Inserting large text files into text widgets results in delays which cannot be fixed by preprocessing, and cannot be canceled. The text widget could better handle large text insertions by occasionally processing idle events. This would allow users to page through the early parts of the text while the later parts were still being loaded in.

**Portability**

In order to reach a large cross section of users on the internet, NR had to be portable to a variety of architectures. Tcl was an ideal language, because the Tcl language is almost entirely architecture and operating system independent. However there were some barriers to portability.

NR requires network socket code to communicate with the NNTP server. Until recently, NR required the C-code extension Tcl-DP[9] to provide these socket commands. A special wish had to be compiled with the Tcl-DP extensions in order to run NR. With the release of Tcl7.5alpha, NR was distributed with a loadable Tcl-DP extension that was dependent on the binary architecture. Now that Tcl7.5beta has support for native sockets, NR will be a Tcl application that requires no extensions to the core Tcl interpreter.

The cross-platform support introduced in Tcl7.5 and Tk4.1 has made NR accessible to millions of Microsoft Windows users on the Internet. However making NR runnable on a Windows machine required considerable modification, due to heavy use of the "exec" Tcl command in some of NR's code modules. Here are some suggestions to writing portable Tcl code:

*Avoid exec whenever possible*. This one should be clear. Programs that exist on Unix may not exist on Windows and vice versa.

If you must use exec, *don't hard-code the commands into each exec*. Use a variable to represent each command, so that you can change your code to use a different command by only changing a line or two. `exec $rmProg $filename` is much better than `exec rm $filename`. On a similar note, don't use "/bin/sh" to start your commands or rely on shell specific substitutions or commands. This mistake has caused me considerable grief.

*Remember that the format of X-defaults is completely foreign to Windows users*. There is nothing wrong with using X-defaults to save your preferences, but don't expect Windows users to edit those files directly.

## Resource Locking in Tcl

NR maintains a single network socket for communication with the server. Since NR allows the user to be browsing multiple groups at once (each in a separate window), access to the network socket must be controlled to prevent multiple groups from accessing the socket simultaneously. Some method of resource locking must be used. Locking is not completely intuitive in Tcl, because only a single execution context is supported. However, there turns out there is a very simple solution to resource locking in Tcl. A single global variable is used to represent the lock and processes desiring to obtain the lock enter a busy-wait loop, using the “after” command. For example, any Tcl procedure needing to access the network socket would have a form similar to the following:

```
Proc GetOverview { group } {
  global socketLock
  if { $socketLock == 1}
  {
    after 500 [info level 0]
    return
  } else {
    set socketLock 1

    # Main code of procedure
    # follows
    ...

    set socketLock 0
  }
}
```

Upon failing to acquire the lock, the procedure uses the after command to schedule itself again in 500ms, in hope that the lock will then be free. `[info level 0]` returns the current procedure name and arguments. This simple form of resource locking works because Tcl has only a single thread of execution and reads and writes to the lock variable are guaranteed to be atomic. Those more familiar with the theory of process synchronization will point out that the method described above is not guaranteed to be fair. Processes waiting for a resource are not guaranteed to be serviced in a first come-first served manner. This was not a requirement in NR.

## Examples of Using NR for Information Filtering

GroupLens is a collaborative-based filtering system that is currently being targeted on Usenet news. A GroupLens-enhanced newsreader provides predictions as to how interesting a user will find each news article. GroupLens works by having users assign ratings to each news article they read based on how interesting they found that article. Ratings are sent to a central GroupLens server, known as the Better Bit Bureau or BBB. The BBB correlates all the users’ ratings and clusters users of similar interests. The BBB then generates predictions for each user based on the ratings of users in the same interest cluster. The newsreader client can then chose how it wishes to use the predictions. It could just display them (as shown in figure 3), sort them, or even chose to display only articles with a prediction above a certain threshold.

Another filtering agent that I am working on is completely content based. This filtering agent requires no explicit interaction from the user, gathering all its information by simply watching the keystrokes and mouse events of the user. When reading news, a user selects potential articles from a list of all available subject lines. The filtering agent works on the assumption that there is some method to the manner in which a user chooses certain subject lines and ignores others. The agent remembers those subject lines chosen for reading as positive examples, also remembers lines that were ignored as negative examples. The agent builds an AI structure


Figure 3: NR with GroupLens predictions. The predictions are represented here as ASCII scales. The larger the scale, the better the prediction.

called a decision tree, based on what keywords signify interesting articles and what keywords signify bad articles. The decision tree can then be used to classify new articles as interesting or uninteresting. Figure 4 shows NR displaying ratings on a A-F scale. Since the displayed ratings are actually menubuttons embedded in the widget, a user can click the mouse on a rating to determine exactly why that subject was assigned a certain prediction as shown in Figure 5. The resulting menu also offers an option to provide feedback as the accuracy of the prediction. This feedback can then be used to help improve the building of the decision tree.

## Current Status of NR

An alpha release of NR is currently available from ftp://ftp.cs.umn.edu/dept/users/herlocke/nr. The extensibility interface and documentation have not been completed yet. Once work on both these issues is completed, a beta test of NR will be announced. If you wish to be added to the announcement mailing list, please send me a email note (herlocke@cs.umn.edu).

## Conclusions

NR has grown from a simple prototype to a large application of around 16,000 lines. The process of evolving NR has been relatively painless because the Tcl language supports and encourages extensibility. I have shared some of my key experiences from developing NR, and I hope that the Tcl/Tk community will continue to develop highly configurable and extensible applications. To restate the key points:

*Tcl/Tk is an excellent language for rapid development of full featured applications*. It's not just for creating prototypes. Performance problems can usually be solved through proper structuring of the application.

*Organize your code in a consistent, modular fashion*. Follow a code organization scheme such as the one described in [10], or perhaps use the object-oriented facilities described in [incr Tcl][4], OTcl[8], or Object Tcl[11].

*Share your useful code*. If you have written a useful code module, don't be afraid to share it with the rest of the Tcl/Tk community. Chances are that one of us would like to use it too. But please document it well.

*Write easily extensible applications*. Ideally, application extensions will be exchanged among users casually, promoting all users to share their labors of customization.

*A runtime constraint system is an excellent way to trig-*

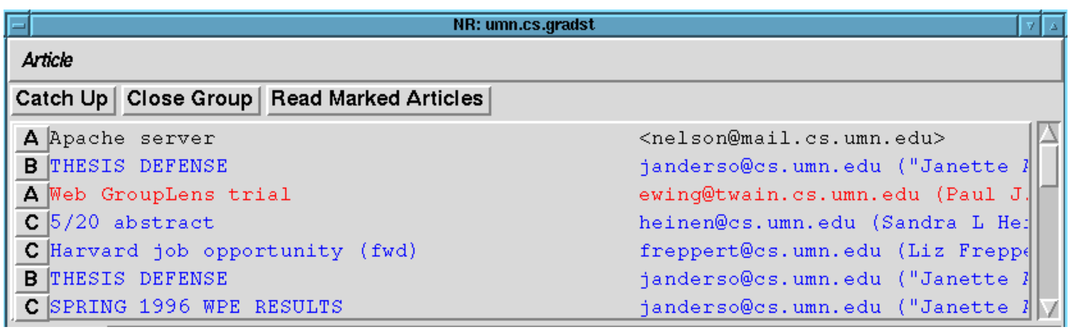


Figure 4: NR with A-F predictions, based on keywords found in the subject line. Note that the ratings are actually menubuttons embedded into a text widget.

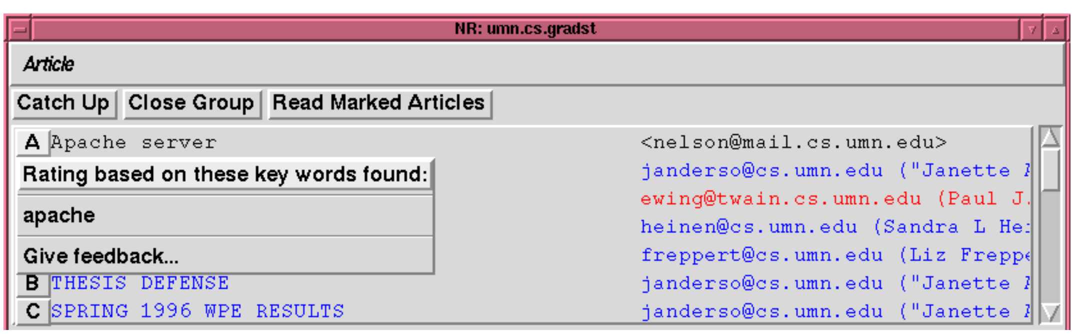


Figure 5: The embedded menubuttons are used both to provide more information about a prediction, but also allow the user the opportunity to provide feedback on a specific prediction.

*ger events based on state changes*. Tcl-Prop works wonders.

*Don't compromise portability with use of exec*. Try and find another way.

## Appendix - Features of NR

- Intuitive GUI interface based on Tk
- True multilevel threading based on "References:" lines.
- Support for display of MIME articles.
- Optimized for use over slow network links such as 14400 modems.
- Support for browsing multiple newsgroups simultaneously
- Highly customizable interface.
- Large shared code base with the popular exmh mail reader allows sharing of code extensions

## References

1. GroupLens. *http://www.cs.umn.edu/Research/GroupLens/*
2. Sunanda Iyengar and Joseph A. Konstan. TclProp: A Data-Propagation Formula Manager for Tcl and Tk. *Proc. of the 1995 Tcl/Tk Workshop*, Toronto, Ontario, Canada. July 1995.
3. Brian Kantor and Phil Lapsley. Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News. RFC 977. *ftp://ds.internic.net/rfc/rfc977.txt*. February 1986.
4. Michael McLennan. The New [incr Tcl]: Objects, Mega-Widgets, Namespaces, and More. *Proc. of the 1995 Tcl/Tk Workshop*, Toronto, Ontario, Canada. July 1995.
5. Brad A. Myers, et. al. Comprehensive Support for Graphical, Highly-Interactive User Interfaces: The Garnet User Interface Development Environment," *IEEE Computer*, November 1990.
6. Ellen Santovich, Rick Spickelmier, and Jonathan Kamens.The XRN newsreader. *ftp://ftp.cam.ov.com/pub/xrn*.
7. Paul Resnick et. al. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. *Proc. of the ACM 1994 Conference on Computer Supported Cooperative Work.* Chapel Hill, NC. 1994.
8. Dean Sheehan. Interpreted C++, Object Oriented Tcl, What next? *Proc. of the 1995 Tcl/Tk Workshop*, Toronto, Ontario, Canada. July 1995.
9. Brian C. Smith, Lawrence A. Rowe, Stephen C. Yen. Tcl Distributed Programming, *Proc. of the 1993 Tcl/TK Workshop*, Berkeley, CA, June 1993.
10. Brent Welch. Customization and Flexibility in the exmh Mail User Interface. *Proc. of the 1995 Tcl/Tk Workshop*, Toronto, Ontario, Canada. July 1995.
11. David Wetherall and Christopher J. Lindblad. Extending Tcl for Dynamic Object-Oriented Programming. *Proc. of the 1995 Tcl/Tk Workshop*, Toronto, Ontario, Canada. July 1995.