Knowledge Server Toolkit: Perl-based Automation Tools

David McLean
AlliedSignal Technical Services Corporation
7515 Mission Drive, Lanham, Md., 20706
301 286 2359
mcleand@pop500.gsfc.nasa.gov

Abstract. A Perl Inference Engine (PIE), A Plan Executor (APE), and a Centroid Classifier (CCL) are described within the infrastructure provided by the Knowledge Server Toolkit (KST) environment. All these tools are implemented as Perl modules so that user's applications can easily use their capabilities. This technology was developed at NASA Goddard Space Flight Center under a task that was looking into languages and tools that are useful for automating satellite ground control systems. However, because these tools are modularized and knowledge driven, this technology should be generally applicable to many fields. After a background description, the Knowledge Representation (KR) and search strategy used by PIE are described. PIE applications are described next and include a simple data checker filter and a PIE Knowledge Server (PIEKS). Then, some simple monitors that display raw and inferred data are also described. These include real-time displays of alpha-numeric data and plots of raw and calculated data. Next, the environment of APE (APE is a client of PIEKS) and its KR are described. A description of a CCL server follows. CCL allows the user to define fuzzy classifier applications and therefore the code should prove useful as a template for writing neural net classifier servers. Finally, the application of KST to automating the Earth Observing System Data and Operations System (EDOS) is briefly described.

Introduction and Background

Knowledge Server Toolkit (KST) is composed of tools that have been useful for automating spacecraft ground systems in the past. Some of these tools are described below.

The Perl Inference Engine (PIE) is a Perl version of a backward chaining search engine called the Transportable Inference Engine (TIE1) [1986] that was developed as part of a study to create new planning and scheduling technology at Goddard Space Flight Center (GSFC).[1] TIE1 was used to do constraint checking in an initial prototype of a reactive scheduler called the Planning And Resource Reasoning (PARR) tool [1989].[2] PARR has since become a tool that supports a number of satellite scheduling applications including the Extreme Ultraviolet Explorer [1992] and the Hubbel Space Telescope servicing mission [1993].[3,4]

The PIE module can be used to create a number of different types of applications. For example it can be used to examine a stream of data and check for accuracy of errors or it can be used in conjunction with a data server to create a knowledge server. A PIE Knowledge Server (PIEKS) can, in turn, serve clients that rely on it to display status or take action depending on the inferred state of things.

A Plan Executor (APE) has evolved from ideas gleaned from PARR, the Generic Inferential Executor (Genie) [1996], human-computer cooperative problem solving research [1995], and experience automating the Gamma Ray Observatory ground system [1998].[5,6,7] Like the original version of PARR it uses an inference engine (now PIE) to do constraint checking (check the preconditions and postconditons of activities). The separation of the constraint checking software from the action component of the plan executor has allowed greater flexibility and reuse of the constraint knowledge as well as creating an independently useful search engine. APE is implemented as a client of PIEKS and therefore uses knowledge stored in a PIE Knowledge Base (KB) as well as raw data served through PIEKS.

A Centroid Classifier (CCL) was also developed to fill the gap left by PIE's brittle, heuristic-based classification approach. CCL is a tool that allows the user to define patterns of data that represent useful diagnostic cases and also "cutoff" values for how close data must be to patterns (centroids) before case membership is assigned.

Overview of the Knowledge Server Toolkit

KST is a set of socket-based applications of monitor and control technology that can be used to solve problems in real-time. All of the tools are written in Perl.

The KST downloaded site is http://lambda.gsfc.nasa.gov and includes the following components:

Basic Infrastructure

1. DSI -- Data Server Interface
2. PIEKS -- Perl Inference Engine Knowledge Server
3. APE -- A Plan Executor
4. CA -- Condition Action Executor
5.CCLS -- Centroid Classifier Server

Applications

1. APEMON -- APE Monitor
2. DMON -- Data Monitor
3. HSMON -- Health and Safety Monitor
4. HSDOC -- Health and Safety Doctor
5. FMON -- Data categorizer and frequency Monitor
6. CCLMON -- CCL Monitor
7. CCLHSDOC -- CCL-based Health and Safety Doctor

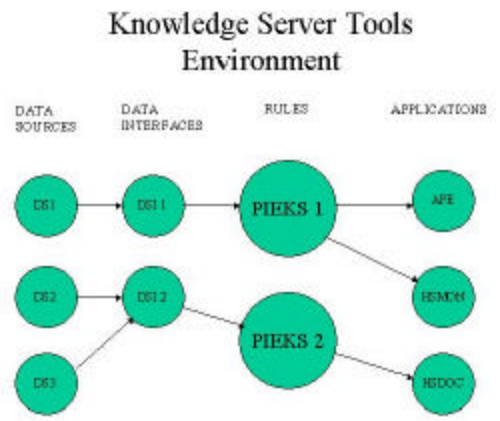

Figure 1 Knowledge Server Tools

DSI connects to raw data servers, PIEKS connects to DSI, and applications like APE and monitors connect to PIEKS. There can be multiple raw data servers, DSI, PIEKS, and PIEKS clients.

DSI is the bridge between external data and KST. It loads data and makes it available for PIEKS clients.

PIEKS is an application of the Perl module PIE.pm that connects to DSI to get current data every N seconds. PIEKS can return raw data, data set by the client or inferred data. If the name of the data requested matches a raw data type than its value is returned. If the reserved word "SET" is used before the data specified and followed by a value then data is set to that value. Inferred data is returned if a KB frame name exists that matches the name sought. When this is the case, the rules are evaluated and the return values are appended to the inferred data to be returned. Thus, the value of inferred data can be multiple as is often the case when multiple things go wrong.

APE is an application of APE.pm that connects to PIEKS to set its parameter data and check the constraints (preconditions and postconditions) of activities. APE also connects to APEMON to highlight the activities that are currently being executed.

CCLS is an application of CCL.pm that also connects to PIEKS to get its data before classification begins. Clients like CCLMON and CCLHSDOC connect to CCLS to get centroid classification data.

A Perl Inference Engine

Requirements for a PIE

A PIE should not only do things that "normal" inference engines do but also should take advantage of the power of Perl. Among the list of things that "normal" inference engines provide are:

1. Powerful knowledge representation structures
2. String and numeric values
3. Default values
4. Setting data from an application
5. Traceability of search strategies.
6. Inference explanation
7. Embedability of engine
8. Search strategies that include:
   a. Examining raw data
   b. Examining known inferred data
   c. Inferring data via backward chaining

Among the list of things that should be added to this list because of the Perl implementation are:

1. Compiled cases that look like Perl statements
2. Numeric calculations of data in real-time
3. Return of data from external executables
4. "Rules" that include pattern matching
5. Light weight and maintainable code
6. Portable code

In retrospect, PIE has evolved into a tool that is really just a set of convenience functions that frame AI type search via the power of Perl.

PIE Knowledge Representation

As with TIE1, a frame is PIE's representation of a chunk of knowledge that represents a class of cases in problem space. Frames include a frame name, a default value, and sets of cases. Each case includes a case name followed

by a list of attributes with specified limits of their values. The general syntax for frames is as follows:

```
Frame_name1
DEFAULT unknown
name1
        attr_name1 relation value
        attr_name2 relation value
         ...
        attr_nameN range v1 v2
.
name2
        attr_name1 relation value
        or
        attr_name2 relation value
        ...
        attr_nameN range v1 v2
.
…
.
Frame_name2
      ...
.
...
```

A frame includes a frame name and a set of cases that represent the possible discrete values for the frame along with rules that define the cases. The frame name is the name of a goal that is being sought. The value assigned to the goal is one or more of the case names. Case names are assigned to a goal when the rules that define them are evaluated as true.

The DEFAULT value slot is optional. This is a way of returning a value when none of the cases match the data. The default DEFAULT value is OK -- This is useful for writing APE constraints where only the problem types need to be specified. An attribute name within a rule may refer to raw data or may refer to another frame name. If the attribute refers to another frame then the attribute is said to be an inferred attribute. Attribute names can be any alphanumeric string, including '_'. A relation may be one of [eq, ne, !~, =~] for comparing strings or [<, <=, ==, >=, >] for comparing numeric values. A range may be one of [ra, nr] (in range, not in range) for checking numeric range values. The set of attribute-relation/range-value triplets that comprise the body of a case is an implicit rule that defines the case. Each slot is connected with an implicit "and" but explicit "or"s may be inserted by the KB author.

If the case name is the reserved word "CALC" then a calculation is expressed on the following line. For example:

```
EM1LDBSV_OVLBEON2
CALC
        EM1LDBSV - OVLBEON2
.
.
```

Here the value of EM1LDBSV_OVLBEON2 will be assigned to the difference between EM1LDBSV and OVLBEON2. Note that '_' was used in the name instead of '-' because things like -+/* are numeric operators that just confuse things if used in attribute names.

A calculation can also be a string returned by another application. For example:

```
CBR_value
CALC
        `cbr $thing`
.
.
```

The backquotes indicate that executable cbr is to be run with the value of $thing passed to it as an argument. The string returned by this invocation is assigned to CBR_value. This is a simple way to get information from other "agents".

When a PIE KB is read, it is compiled into strings that represent logical statements (rules) that are evaluated at runtime during search. This is nice because instantiated rules can also be returned that serve as explanation for why inferences were made. A PIE KB is represented in Perl by a hash of frame names each of which contains sets of case names (another hash) representing the problem space. This is a nice way of packaging and "indexing" the knowledge for later search because hash lookup is fast.

PIE Search

PIE represents raw data by attribute names (Perl variables) that have been assigned values via the "updata_data" API. During search, PIE first looks for raw data values, then for known inferred values, and finally it backward-chains through the frame hierarchy to infer the value from known values in rules. Because PIE "remembers" inferred values during an inference session, it does not waste time searching and evaluating similar inference rules again. When data is being served by PIEKS and new raw data becomes available, the inferred values hash is cleared so that normal search is resumed. During inferencing, the names of all of the case rules that evaluate as true are returned and assigned to the inferred value. This is useful when many different things can go wrong at the same time.

Other Features of PIE

The PIE module, infer, can be called with a SET command to set some data. This is useful to set some initial parameter data before more "real-time" inferences are made because it allows the KB author to write case frames with user set data as part of the rules. For example, AOS_time can be set before a spacecraft pass to instantiate the AOS time from a scheduling system before a real-time pass.

PIE will print a search trace if passed the switch "-t" and will return "explanation" through the explain API if passed the "-e" switch.

PIE comes with a simple WWW forms-based editor to add, delete, and edit PIE KBs.

PIE Applications

A Spreadsheet Checker

The following example is presented here to give the reader a feel for the code and how some of its components work. One of the simplest types of applications of PIE is that of range and limit checking. For example, a PIE KB can be defined that expresses the conditions for when values are out-of-bounds in a spreadsheet:

```
check
CDB_RCVD_BAD
        CDB_RCVD      nr      2 6
.
CBDS_XFER_GT_5
        CBDS_XFER     >       5
.
INV_CDB_FINE
        INV_CDB       ra      0 1
.
ALL_ZERO
        seq_cdb_check eq      SEQ_CDB_ZERO
        INV_CDB       ==      0
.
.
# inferred value:
seq_cdb_check

SEQ_CDB_ZERO
        OUT_SEQ       ==      0
or
        CDB_RCVD      ==      0
.
.
```

In this example the major inferred data name, or goal, for PIE inferencing is “check”. All of the data is numeric and the first 3 cases express simple limit-checking rules. The last case “ALL_ZERO” uses an inferred attribute, “seq_cdb_check” in its rules which causes PIE to infer that value from the frame below.

The complete script required for this type of application is given below:

```
#!/usr/local/bin/perl
# pie.pl

use PIE;
# get flags and goal name
while(($goal = shift) =~ /-/) {
        $flags .= $goal;
}
# get KB name
$kb = shift;
# get data names
chop($N = <>);

$p = new PIE($flags,$kb,$N);
$p->print_kb();

# process lines of data
$n = 1;
while(<>) {
        chop;
        $p->update_data($_);

# seek goal
        $results = $p->infer($goal);
        $n++;
        print "$n $results\n";
        $results = $p->explain();
        print "$results\n" if $results;

# delete old inferred facts (and explanations)
        $p->clear_ifacts();
}
```

This script assumes that file (or data stream) being processed is in PIE's "table-speak" format -- tab delimited with the first line containing the names of the data attributes (like a database table). Pie.pl is invoked like "pie.pl -t check sw.kb sw.dat" and then gathers the flags (-t == trace and -e == explain) and then the goal (frame name, ie. check) to seek and the KB name (sw.kb). Next, the first line of data is read (from file sw.dat) to get the data names and then an instance of PIE is created by passing it the flags, KB name, and data names. Then, for debugging purposes, the compiled KB is printed (via print_kb). Next, the script processes each line of the file, setting values of the attributes (via update_data) and

returning the inferred value of the goal based on that line of data. The line numbers are printed so that any out-of-bounds cells can be easily located within the spreadsheet. If the - e switch is passed to the script then this application will also print an explanation (based on the KB definitions). Finally, the inferred facts and any explanations are cleared before the next line of data is processed.

GSFC's Flight Dynamics Facility has used this approach to identify problems in their quality assurance reports.[8]

PIE Knowledge Server

PIEKS is a socket-based application that connects to a data source through DSI, loads a PIE KB and then serves clients who request the values of data. The data requested by clients is expressed in terms of the data names or goal names (inferred data). Clients do this by connecting to the PIEKS socket and sending a string with one or more data/goal names. PIEKS returns the respective values for raw data requested and all the values of inferred values requested. PIEKS also returns explanation in terms of uninstantiated and instantiated rules if requested by the client. Clients may also request that data be set to specified values by using the reserved token "SET" before the data name and value.

PIEKS Clients

Monitors

The following monitors are simple, template-like GUIs, written in TkPerl, that are meant to be starting places for more sophisticated monitors. The emphasis of tools like PIE and APE is automation without human intervention -- If the task is scripted right then there should be no need for human interfaces.

APEMON highlights buttons with activity names that represent the plan being executed. The button color indicates the state of the plan execution, for example, green indicates a completed task and red indicates a failed task.

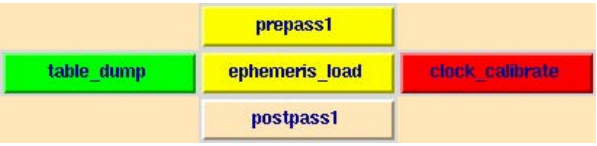

Figure 2. APE Monitor

DMON connects to PIEKS and plots the values of data in real time.

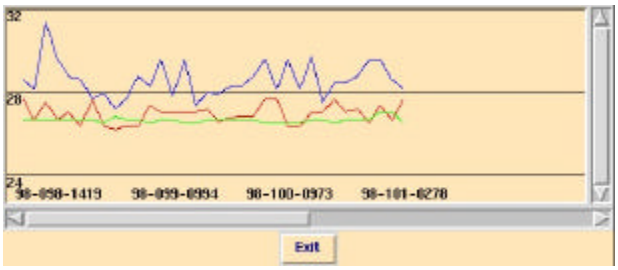

Figure 3. Data Monitor

HSMON is a simple Health & Safety (H&S) monitor included with the PIE distribution. It connects to PIEKS every N seconds and updates a display of the H&S status. HSMON can easily be configured for use by passing it parameters. For example, the following invocation

```
hsmon.pl +0 +0 10 gro `hostname`&
```

configures hsmon.pl to display the values returned for the frame "gro" every 10 seconds in a display that starts at x position +0 and y position +0.

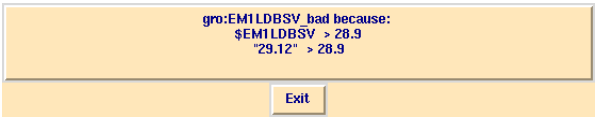

Figure 4. Health & Safety Monitor

HSDOC connects to PIEKS, as HSMON, but also launches APE whenever a problem is identified that can be fixed.

FMON connects to PIEKS, as DMON, and classifies the data into N bins of size S about a given value and then displays a histogram of the bin counts every N seconds.

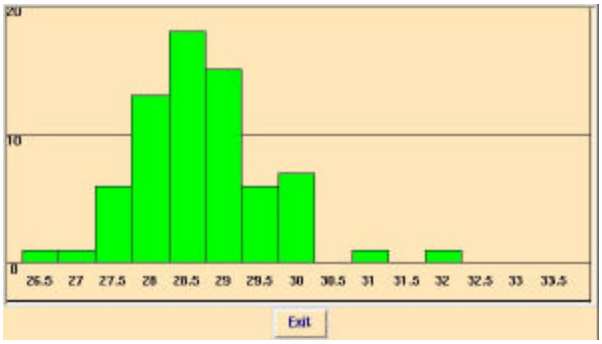

Figure 5. Frequency Monitor

CCLMON connects to CCLS and displays the real-time centroid among the diagnostic case centroids.

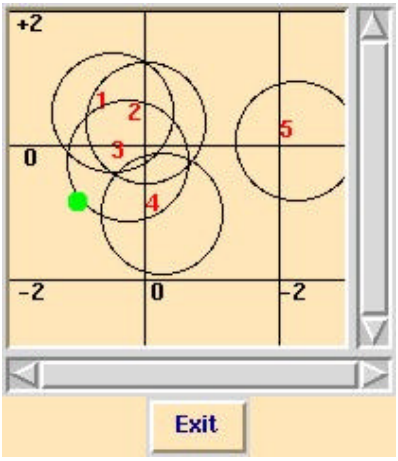

Figure 6. Centroid Monitor

CCLHSDOC connects to CCLS and launches APE applications, with the intent of fixing the problem, whenever a diagnostic case centroid is returned.

Figure 7. Centroid Health & Safety Doctor Monitor

A WWW Proxy PIEKS

It is easy to write a CGI script that connects to PIEKS to get current raw and inferred data and then load itself every N seconds. The returned data is then put in HTML tables and returned to the WWW client.

By using JavaScript to extend this script, one can also implement an "alarm page" that acts as a log of all the inferred data that had problems as well as explanations.

With a little more effort a CGI can be written that invokes a Java Applet that displays a graphic timeline of data that is updated every N seconds.

A Plan Executor

APE.pm is a Perl module that can be used to write plan executor applications that connect to PIEKS to do constraint checking. Activities and plans are defined in separate files. An activity has the following format:

```
Activity_name1
        precon1 t1, precon2 t2, ...
        activity_script N
        postcondition t1
        alt_activity1 alt_activity2 ...
.
Activity_name2
    ...
.
...
```

Preconditions and postconditions are referred to by PIE frame names. The t1, t2, etc. are the respective timeouts (in seconds) for each precondition and postcondition. Activity_script is the name of the script to be executed when the preconditions are passed. The N after the activity script name is an optional retry count that is in force if the postcondition fails. Activity scripts can be any executable script, in any language and normally should write to a log file. Optional Alt_activitys can be used when the primary (current) activity fails.

A plan file is just a list of activities to be executed. Plan generation software might automatically generate this file:

```
Activity1
Activity2
Activity3 Activity4 Activity5 ...
Activity6
...
```

When more than one activity is displayed on a single line, these activities are executed in parallel. All of these activities must complete before executing any of the following activities. Plans can be nested so that hierarchical plans can be defined by including plan file names in a plan. APE also reads an optional parameter file when it starts. The parameter file is used to set initial values of things like orbit number, AOS time, EOS time, and configuration codes.

As activities are processed, APE sets state variables that match the activity names, setting them to "START", "END", and "BAD" as the activities are started, ended normally or ended with error. This gives further control of the activities by referring to the state variables in the constraint rules.

APE is invoked by typing the application name (ape.pl) followed by the name of an application type. The

application type given, for example "gro", tells ape to look for a gro.kb (activity KB), a gro.parm (optional parameter file), and a gro.plan (plan file). Ape may also be invoked with a -t (trace) switch that displays the details of its plan execution.

APE comes with a simple WWW forms-based editor to add, delete, and edit frames within an APE activity file or plan file.

Condition Actions Applications

There may be many cases when the automation application is very simple and does not need all the reactive plan execution features that APE has. When this is the case it is possible to use the much simpler tool, called the Condition-Action (CA) application. CA uses a condition-action list to control its behavior. The conditions are expressed in terms of PIE inferred values and the actions are expressed in terms of the names of scripts to be executed when its paired condition is found.

The PIE distribution comes with examples of CA applications that connect to user-defined data sources and to PIEKS.

Centroid Classifier

CCL.pm is a Perl module that is designed to write Centroid Classifier Servers. These servers first read a centroids file that contains the means and sigmas for the data cases being used in the classification problem. The centroid server then connects to PIEKS to get the data required and serves clients whenever they make requests. A fuzzy classifier is useful for identifying diagnostic categories when the data is not a precise predictor of the diagnostic categories.

The centroids used by the classifier represent patterns that identify the mean profiles of diagnostic cases defined by experts or discovered by an empirical cluster analysis. A typical classifier application will define a "cutoff" that defines the furthest distance from a diagnostic centroid that a data centroid can be without being assigned to a centroid group. For example, if the cutoff is defined as one standard deviation than the data centroid must be within one standard deviation of a centroid to be assigned to that diagnostic case.

The centroid classifier server that comes with the distribution of CCL.pm always returns the closest centroid if within the cutoff or it returns the value of "OK" to indicate that the data is "normal". This application always normalizes the data before making a classification. The list of applications that come with the CCL.pm distribution are as follows:

ccls.pl
Centroid Classifier Server

cclhsdoc.pl
client of ccls.pl that displays the centroid returned and then launches an APE script to take care of things.

cclmon.pl
display that plots the case centroids in the plane of the first 2 data attributes used in the classification and then plots the data centroid in real-time.

Automation of the Earth Observing System Data and Operations System (EDOS)

NASA's Earth Observing System (EOS), formerly known as NASA's Mission to Planet Earth, objective is to study the Earth via a series of polar-orbiting and low inclination satellites over the next two decades. The ground operations segment of EOS, known as EDOS, has been tasked with automating as many of its tasks as possible.

KST technology is especially easy to use if the environment where it will be used is already using Perl to automate tasks. This is because when the Perl Automation culture is in place, it is easier to think about how to extend and enhance the automation by using other Perl-based tools. This is the case with the EDOS ground system team which is using Perl to format data and implement CGI scripts that display dynamic HTML pages that present data in tables and as graphic displays.

The EDOS team is continuously looking for opportunities to automate the ground system. This effort has lead to the development of a prototype that uses KST technology to monitor critical telemetry data at 10 second intervals through a proxy PIEKS and display the data values in HTML tables. Figure 8 shows the tables of data that are displayed for normal data. The PIE KB behind this display expresses the limit checking and other heuristics that indicate when data is anomalous. When bad data is found, JavaScript is executed and launches a log window that lists the date, time, and values of data that are anomalous. Explanation is also provided in the form of PIE rules that have concluded that the data is bad.

Another prototype displays EDOS real-time data in a timeline plot via an applet. Figure 9 shows part of the display presented by connecting to several applets. Each applet connects to a proxy PIEKS and updates its display every 10 seconds.

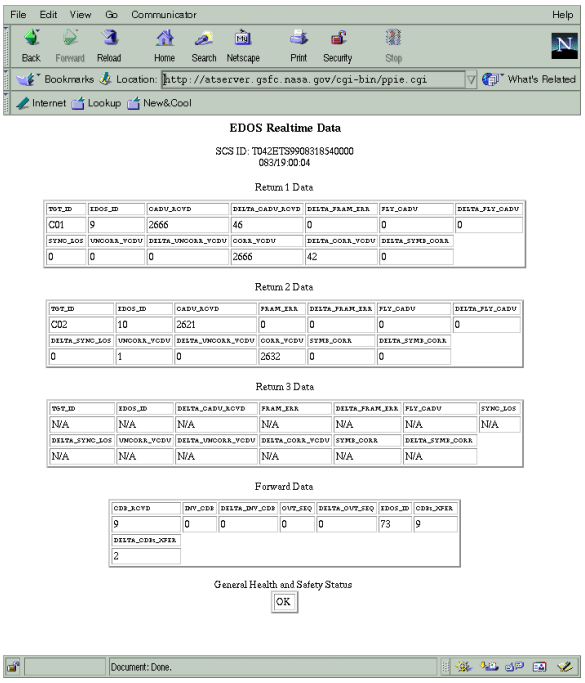

Figure 8. Real-time Data Display

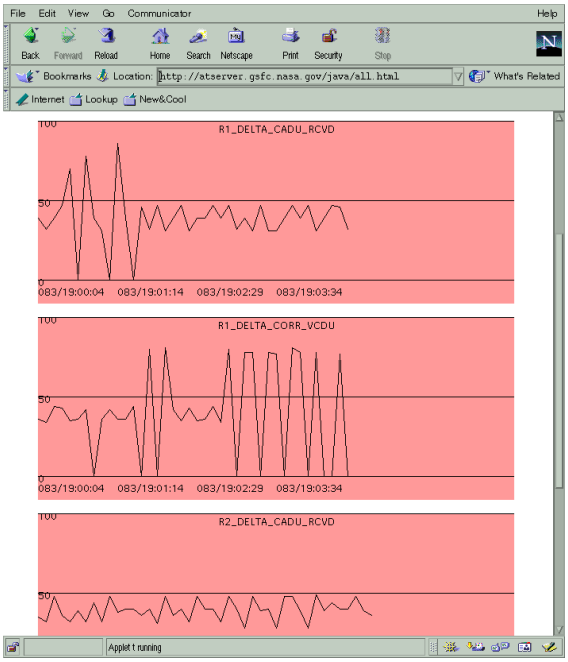

Figure 9. Real-Time Data Plots

This EDOS monitoring system will be extended to include a paging system that will send pages to engineers when anomalies are found. The paging system uses automation technology developed for automating the GRO and XTE ground systems. This system consists of a Perl script that configures Kermit and then connects to a ROLM digital phone and then to a modem through a serial port on a workstation. Pages for the EDOS RT monitoring system will be initiated by a Condition-Action script that uses a PIE KB to express critical conditions and executable scripts that page different groups of engineers, depending on the type of anomaly found. There are potentially many chances to use this type of application throughout the EDOS ground systems.

In general, KST applications may include any routine activities that can be identified where data is available (or can be made available) so that conditions can be expressed in the form of PIE KBs and actions can be expressed in the form of scripts that execute when these conditions are found. Such routine tasks as report generation and anomaly detection have already been described. Future tasks to be automated will include automatic execution of routine remediation scripts when anomalies identified.

Summary and Conclusions

The concept of a knowledge server greatly enhances the power of this set of tools. Connecting to a knowledge server allows clients to get raw data, inferred data, and user data (data set by the client). KST is written in freeware Perl which is a very powerful and fast scripting language. One language is used to build everything so maintenance is simplified. The tools communicate with sockets so each component can be running on a different workstation.

The knowledge representation used by PIE is a natural one to represent model-based reasoning because frames can be organized into hierarchies. The knowledge representation used by APE separates plan, activity strategy, and constraint checking into three simpler and more reusable files. For example, different activities can use the same constraints and different plans can use the same activities. The CCLS tools allows users to classify fuzzy problems and use the result to display the results or do something about the problem. The monitors provided by KST are simple, fully working monitors that can be extended as a user sees fit.

The author envisions fully autonomous systems, based on reactive planning and scheduling technologies, that automatically select engineering and science goals, generate plans, and then execute the plans that achieve those goals. KST is a step in the direction of realizing this goal. As more hardware resources (memory and processing power) become available on-board spacecrafts and better software (plan generators) become available, these systems will move from ground-based autonomy to spacecraft-based autonomy.

Acknowledgements

The author wishes to thank Mark Stirling and Barbie Brown under NASA/GSFC contract NAS9-98100 for their support of this effort.

References

1. McLean, D.R., "The Design and Application of a Transportable Inference Engine (TIE1)," Telematics and Informatics, Vol. 3, No. 3, Pergamon Press, 1986.

2. McLean, D.R., Yen, W. L., "PST & PARR: Plan Specification Tools and a Planning and Resource Reasoning Shell for use in Satellite Mission Planning," Proceedings of the 1989 Goddard Conference on Space Applications of Artificial Intelligence, GSFC, Greenbelt, Md., May 1989.

3. Potter, W., and McLean D., "Explorer Platform Planning System (EPPS): A Mission Operations Planning and Scheduling Tool," GSFC Research and Technology Report, 1992.

4. Bogovich, L., Johnson, J., Tuchman, A., McLean, D., Page, B., Kispert, A., Burkhardt, C., Littlefield, R., and Potter, W., "Using AI/Expert System Technology to Automate Planning and Replanning for the HST Servicing Missions," Proceedings of the 1993 Goddard Conference on Space Applications of Artificial Intelligence, GSFC Greenbelt, Md., May 1993.

5. Hartley, J., Luczak, E., and Stump, D., "Spacecraft Control Center Automation Using the Generic Inferential Executor (Genie)," Proceedings of the SpaceOps 96, Munich Germany.

6. Jones, P.M., and Mitchell, C.M., "Human-Computer Cooperative Problem Solving: Theory, Design, and Evaluation of an Intelligent Associate System," IEEE Transactions on Systems, Man, and Cybernetics, 25, 7, 1995.

7. McLean, D., Zhang, Y., and Fatig, M., "Automating the Compton Gamma Ray Observatory Ground System: Expert System and WWW Technologies," Proceedings of the World Congress on Expert Systems, Mexico City, Mexico, January 1998.

8. Automation Tools for Identifying Problems in QA Reports, GSFC Technical Report, January 1998.