AUTOMATION TOOLS FOR ACCELERATOR CONTROL A NETWORK BASED SEQUENCER

Peter Clout, Mark Geib and Robert Westervelt

Vista Control Systems, Inc., 127 Eastgate Dr. #30800, Los Alamos, NM 87544

Abstract

In conjunction with a major client, Vista Control Systems has developed a sequencer for control systems which works in conjunction with its realtime, distributed Vsystem database. Vsystem is a network–based data acquisition, monitoring and control system which has been applied successfully to both accelerator projects and projects outside this realm of research. The network–based sequencer allows a user to simply define a thread of execution in any supported computer on the network. The script defining a sequence has a simple syntax designed for non–programmers, with facilities for selectively abbreviating the channel names for easy reference. The semantics of the script contains most of the familiar capabilities of conventional programming languages, including standard stream I/O and the ability to start other processes with parameters passed. The script is compiled to threaded code for execution efficiency. The implementation will be described in some detail and examples will be given of applications for which the sequencer has been used.

The Problem

Most control systems provide a conventional computer programming interface for the operators and users to aid in the automation of the system as the experience, understanding and confidence in the operation of the machine increases. The addition of these programs to the control system is vital to the continued development of understanding and performance up to and sometimes past the design goals. Several attempts have been made in the past [1,2] to make this process easier for the users and operators. These attempts have focussed on developing interactive languages with realtime and multi–processor enhancements. This has resulted in facilities that have been easy to use for debugging programs but where the source program has almost always been interpreted by the computer, resulting in slow execution, and involving the user in defining the I/O to the hardware and taking account of the networking between computers. Thus, the user has had to worry about more than the the basic problem at hand.

Ideally, the user should be able to express the trial solution to the problem in terms that are confined to the language of the problem and which exclude any aspects of the I/O subsystem, the computers, their operating systems and the networking of the computers that make up the whole control system. Together with a client, we have developed an automation tool that we believe advances the art in this area.

The Background for the Line Sequencer

The predecessor to Vsystem was developed at Los Alamos National Laboratory [3,4] and licensed to Vista Control Systems. We have developed and documented the control system, Vsystem, so that it is now a useful product in the marketplace. The backbone for Vsystem is the networked realtime database. This database is used initially to model the I/O connections to the control system. The contents of the database are defined by a regular ASCII file which can be generated directly with editors or with a traditional database. Database channel names can be freely chosen to be generally meaningful. Code then has to be developed to make the connection between the database and the I/O subsystem. This code could well come with the package, but the user is also given the documentation and help to write custom code so that almost any I/O system can be supported. Generally, this code is just a few lines for each type of I/O module. The system also supports any user written data conversion routine that can be expressed in a computer programming language and how to write and include these routines is documented.

The data in the database is accessed through a library of subroutines, Vaccess. The actual database in a system can be distributed among the computers of the system and each computer can have many individual databases installed. Each individual piece of the database is held in main memory as a global section to give good performance. Vaccess routines make the database appear to the user as one database taking care of the computer network access. The concept is that the user interacts directly with the database model of the machine. The code running between the database and the I/O subsystem ensures that the database reflects the state of the actual machine and that requested changes are written to the I/O subsystem.

The second component of Vsystem is the operator interface tool, Vdraw. This tool allows for the rapid development of a graphical interface between the user and the database. It is based on the computing concepts developed at Xerox Parc in the '70s and first popularized by Apple Computers. This tool allows the user to draw operator windows with a draw package and also put objects on the screen which are connected to the data in the database. This allows the operator to quickly see the state of the machine and make changes to settings. By clicking with the mouse in a defined area, the operator can also execute other programs or start another operation in the computer. Vdraw runs under Xwindows and with this tool, complex but intuitive windows can be defined in hours.

Vaccess and Vdraw together provide the tools necessary to build a supervisory control system. They also provide the basis for the extension of the system into automation.

The Line Sequencer

The requirement for the Line Sequencer was to provide a facility whereby scientists not familiar with conventional computer programming languages can define the automation of their experiments, including the analysis of the data, without the need to learn FORTRAN. This language was developed together with a client experienced with this requirement.

The Sequencer provides transparent network management of sequences of action. A powerful feature of the Line Sequencer lies in the fact that since no special programming knowledge

is necessary, the people most connected with the process, the engineers, operators and users, can write the script used to execute action. This eliminates the difficult communications/ specifications step between the user and the programmer.

The three primary components of the Line Sequencer are the compiler, the command interface, and the engine. Commands are entered into the Line Sequencer with a simple English–like language. The **compiler** must translate this English–like script into an intermediate language for the sequence engine. The **command interface** controls and monitors all sequences of actions and communicates back and forth with the engines during user command execution. The **engines** execute source code generated by the compiler.

The Line Sequencer creates an environment which allows multiple servers to run multiple engines, each performing different sequences of actions. Sequences can be repetitive or conditional in nature. Each sequence runs in a separate engine. This allows one server to run multiple engines simultaneously, thus permitting several sequences of actions to execute at the same time. Figure 1 is a diagram of the Line Sequencer environment.

Compiler

The **compiler** is designed for easy use in writing Vsequence script by someone with little or no programming experience. Vsequence Script is based on a simple keyword/value syntax, with all extra text ignored. Because you can include text and keywords simultaneously, Vsequence script is even readable to someone unfamiliar with it. Vsequence script has some of the flexibility and generality of pseudo–code in that there are many ways to express the same action. Also, the sequence compiler generates helpful error messages and will flag conditions or statements that it finds ambiguous. This allows the user several ways of expressing functionality with the confidence that the compiler is interpreting the instructions correctly.

Vsequence Script

English–like constructs can be used in the Vsequence script with keywords used to generate the Vsequence source code. For example, lines of script such as

Turn VALVE12 ON to start the flow of water

or

Initialize by setting VALVE12 ON

are both acceptable. This will set the database channel VALVE12 to the ON state. The extra text allows the script to be self–documenting.

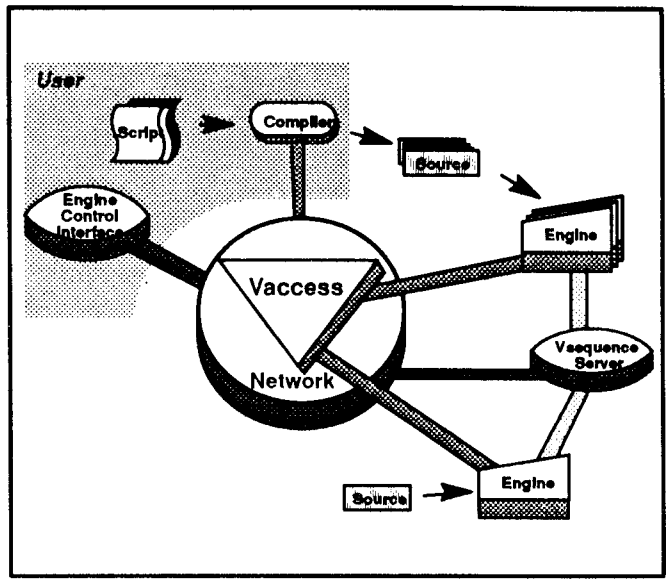

Figure 1: Line Sequencer Environment

The user enters English–like script, which is converted to source by the compiler. This source is then executed by the engine. The Vsequence server monitors and controls the sequence of actions being executed by the engine.

The Vsequence script language has a complete set of flow control constructs for looping and condition testing. In addition, it has a WAIT UNTIL control construct which is useful for holding execution until some condition is reached in the system under control. The WAIT UNTIL control function also has a TIMEOUT capability built–in so that it will not hang while waiting for some condition that may never occur. All of the functionality of a true programming language has been provided. The Line Sequencer's English–like script is a free form, context sensitive, programming language aimed at laboratory automation. It is tightly coupled to the rest of the Vsystem, which will greatly simplifies the command interface.

The Line Sequence Compiler has a rich set of logical operators (expressible in many different forms), mathematical functions, and file I/O capabilities. Additionally, the compiler understands the structure of the Vaccess database and handles setting values in the database transparently.

Command Interface/Vsequence Server

The **command interface** which controls the actions of sequences is based on a client/server model. The *client* is the user interface program which accepts commands from the user and sends them for execution to the appropriate server. The *server* provides and controls the resources necessary for sequence execution, i.e. the sequence engines. This approach allows the user to run sequences without having direct access to the computer on which the sequence is executing, while providing a clean mechanism for distributing the workload over the network.

Each sequence server can support up to 32 engines running on a node at one time. The actual number of sequence engines executing may be less because of the VMS system

configuration. With these 32 engines, the server can also support up to 32 links to users at the same time. Each user's commands are executed in the order in which they are received by the server.

The server also provides security protections for sequences. There are two levels of security for each sequence engine. The engines are run under VMS user accounts and users must have network proxy to access the a specified account. This means that only certain users can run sequences under a given account. Also, only these users can issue commands to the sequence engine. The ATTACH command further restricts access to an engine to one user at a time. Thus, two different users cannot issue conflicting commands to an engine at the same time.

The command interface allows complete control over the creation and execution of sequences. There are commands to create, set the parameters, and show the status of sequence execution. There are also facilities provided for rudimentary debugging of the sequence. The **DEBUG** mode displays the currently executing Vsequence source instruction on the user's terminal, while a single **STEP** mode executes single instructions at a time.

Engine

The **sequence engine** provides an environment which will execute **Vsequence source**. The instructions are intended to provide all the functionality required to write programs which will easily access Vsystem realtime database channels.

The instructions provide for conditional branching, time based operations, limited control of VMS processes, and transparent access to Vsystem database channels. The engine provides limited aid for debugging Vsequence source through a debug mode and single stepping through Vsequence source. C–style I/O is provided and supports any standard VMS stream device. In addition, support is provided for opening DECterm windows on workstations running DECwindows. The engine is stack based; thus, most Line Sequencer instructions take operands off the stack and leave results on the stack. Condition handling is provided by establishing condition handling branches which are taken when a condition occurs at run time.

The execution of Vsequence source is very efficient. The instructions are compiled, and a vector table of instruction entry points provides execution of instructions at run time.

The engine provides a number of utility commands to aid the user. For example, during the loading of Vsequence source the actual source lines for each instruction are stored and can be examined with the DUMP command. The status of the engine can be examined with the SHOW command, which provides the current program counter and status bits.

Example

Figure 2 shows an example Line Sequencer script. In this example, an oven is being ramped up to a given values and then allowed to cool. The "!" character is the comment delimiter and the "load" command causes the sequence engine to search the database defined and load the channel names that fit the search string. In this case, all channel names will match. The channels can now be referenced in the script using only the remainder of the name. In the third line, the words "make" and "degrees" are ignored and in the next to last line there are four words that are ignored.

```
!load all the database channel names
#load FDB::
make set_point 20.0 degrees
!ramp the furnace to 1000
until (set_point > 1000) then
 set_point = set_point + 50.0
 wait 5.0 seconds
end
!wait for sample to reach 1000.0 degrees
wait until ( temp > 1000.0 )
!hold at 1000.0 for 1 min.
wait 60.0 seconds
!go back to room temp
return the set_point to 20.0 degrees
wait until ( temp < 50.0 )
```

Figure 2: An Example Line Sequencer Script

The source will be loaded to the engine for running. Note that the parsed script forms the initial comments and that the engine is a stack–oriented device.

Conclusion

The Line Sequencer is currently in production use and has already demonstrated its power in several demonstrations and a production environment. As the source interface is defined in the documentation, users can develop their own custom interfaces should they wish. We will be developing the capability of the Line Sequencer and also using the engine as the basis for future automation tools.

Acknowledgements

We acknowledge the many interactions with John Thurtell, Greg Maier and Bill Lucas of Mobil Oil Central Research Laboratories in the definition stage of the Line Sequencer. Greg Maier and Bill Lucas also wrote the Script Compiler of the Line Sequencer.

References

1. M.C. Crowley–Milling and G.C. Shering, The Nodal System at the SPS, CERN 78–08
2. P. Haskell, GRACES Reference Manual Version–2, RL–84–043
3. M.A. Fuka P.N. Clout, A.P. Conley, J.O. Hill, R.B. Rothrock, L.L. Trease and M.E. Zander, Proc. 1987 Particle Accelerator Conf. IEEE 87CH2387–9, pp. 652–654
4. P. Clout, R. Rothrock, V. Martz, R. Westervelt and M. Geib, Past, Present and Future of a Commercial, Graphically Oriented Control System, Proc. Int. Conf. on Accelerator and Large Experimental Physics Control Systems, NIM A293 (1990) 456–9