

A SOFTWARE ARCHITECTURE FOR AUTOMATING OPERATIONS PROCESSES

KEVIN J. MILLER

Operations Engineering Lab
Jet Propulsion Laboratory, MS 301-345
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109-8099

ABSTRACT

The Operations Engineering Lab (OEL) at JPL has developed a software architecture based on an integrated toolkit approach for simplifying and automating mission operations tasks. The toolkit approach is based on building, adaptable, reusable graphical tools that are integrated through a combination of libraries, scripts, and system-level user interface shells. The graphical interface shells are designed to integrate and visually guide a user through the complex steps in an operations process. They provide a user with an integrated system-level picture of an overall process, defining the required inputs and possible outputs through interactive on-screen graphics.

The OEL has developed the software for building these process-oriented graphical user interface (GUI) shells. The OEL Shell development system (OEL Shell) is an extension of JPL's Widget Creation Library (WCL). The OEL Shell system can be used to easily build user interfaces for running complex processes, applications with extensive command-line interfaces, and tool-integration tasks. The interface shells display a logical process flow using arrows and box graphics. They also allow a user to select which output products are desired and which input sources are needed, eliminating the need to know which program and its associated command-line parameters must be executed in each case. The shells have also proved valuable for use as operations training tools because of the OEL Shell hypertext help environment.

The OEL toolkit approach is guided by several principles, including the use of ASCII

text file interfaces with a multi-mission format, Perl scripts for mission-specific adaptation code, and programs that include a simple command-line interface for batch mode processing. Projects can adapt the interface shells by simple changes to the resource configuration file. This approach has allowed the development of sophisticated, automated software systems that are easy, cheap, and fast to build.

This paper will discuss our toolkit approach and the OEL Shell interface builder in the context of a real operations process example. The paper will discuss the design and implementation of a Ulysses toolkit for generating the mission sequence of events. The Sequence of Events Generation (SEG) system provides an adaptable multi-mission toolkit for producing a time-ordered listing and timeline display of spacecraft commands, state changes, and required ground activities. The multi-mission SEG software is easily adapted and OEL Shell templates are built to meet different mission requirements. The SEG system was adapted in a unique way for the Ulysses mission since the spacecraft does all commanding in real time. The Ulysses SEG toolkit allows a user to interactively build commands on a timeline display in spacecraft event time and then the system automatically derives required ground events, builds a mission sequence of events listing, and outputs a space flight operations schedule.

INTRODUCTION

The Operations Engineering Lab (OEL) at JPL has developed a generic set of tools for Sequence of Events Generation (SEG) that have been adapted to many of the current

flight projects. The toolkit includes what-yell- SCC-I S-W]; II-y(>II~C((WYSIWYG) editors for the Sequence of Events (SOE), Space Flight Operations Schedule (SPOS), and Deep Space Net Schedule (DSNS), a set

of servers to enhance the Perl language which is used to generate the SFG products, and a user-configurable graphical user interface (GUI) to control the SFG process. All of the SFG interfaces are text files.

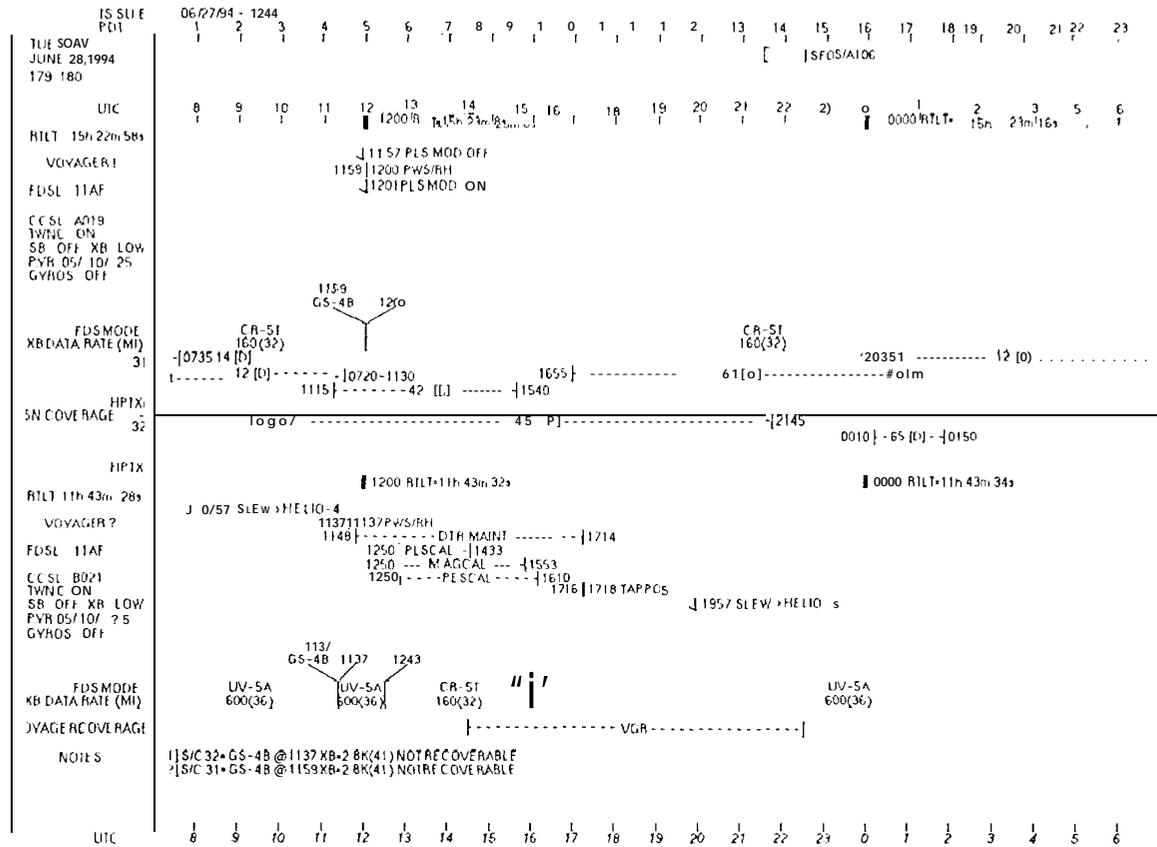


Figure 1. Voyager SPOS

The editors are generic Object-oriented programs that display, edit, filter, and reformat the SFG products, but do not interpret the data. The editors are X Windows / UNIX programs written in C. The same editors are used by all projects. Rather than writing MSDOS or Macintosh versions of the editors, we export files that may be read with most MSDOS or Macintosh tools.

The SFG prompts for most missions is to take the spacecraft sequence file, the DeepSpace Network (DSN) allocations and view periods files, and the light time file, and generate the SOE, SPOS, and DSN keyword files. Simply, SFG integrates the spacecraft and

ground schedules into a unit. The spacecraft sequence file is generated far in advance, does not include real time commands, and is often based on out-of-date DSN allocations. The SOE, SPOS, and DSN keyword files will contain more accurate information, and are used by the Mission Control Team and the Spacecraft Teams to schedule ground activities. In addition all SFG products use ground times for both ground and spacecraft events.

We chose to write our generating software in Perl since it is a very powerful interpreted language designed for processing text files. We also did not want to write a new language. Since the delivered executable is

also the source code, it is reasonably easy for the Mission Control Team to maintain the SIG adaptation. Perl has only two elementary data types: strings and floating point numbers, so additional servers were written in C to manipulate triggers, time-dependent state variables, time conversions, and spacecraft command string processing. The parent Perl script includes a Perl library that automatically starts up the server process and sets up a communications channel between the parent Perl script and the server similar to the Remote Procedure Call (RPC) mechanism. The server functions are then invoked with simple Perl function calls. It is possible to compile new functions directly into the Perl language, but the server model was chosen since it simplifies configuration management on the operations workstations, a new version of Perl may be installed without having to link in any SIG code, and in fact, the servers are not even tied to Perl.

The final component of the SIG toolkit is the OEL Shell. This is a user configurable GUI that lets the user gather input files, specify output files, and selectively run portions of the generating process and the SIG editors. OEL Shells have been built for several projects' SIG processes

THE OEL SHELL

The OEL Shell is a compiled program based on the X11 release 5 windowing system, the X toolkit (Xt), the Motif Widget set, and David Smyth's Widget Creation library (Wcl) [1]. The intent was to provide a shell that would allow the user to enter UNIX commands with parameters from a simple Motif interface. The interface is configurable by the user by modifying the resource file. Several copies (which should in fact be links) of the compiled program may be available on the sys[cm]. The appearance of these shells is determined by the program's name and its corresponding resource file. Since the user is encouraged to modify the resource file, and create one's own shells or enhancements to existing shells, some knowledge of Motif widgets and the rc.source database is prerequisite.

From a user's perspective, the OEL Shell consists of a series of push buttons, text entry areas, and toggle buttons arranged on a work area or control panel (One or more Motif drawing areas). Pressing one of the push buttons causes a UNIX program to execute. This program may be another Motif application or a script without a graphical user interface. The work area provides text entry areas for the user to enter command line arguments for the program. Toggle buttons correspond to UNIX command line options.

Below the pane] is a scrolling message area which displays any output messages from the executing program or script. In addition, the actual UNIX command created from the push button, text, and toggle buttons may optionally be displayed here. If a text widget is used for file input, it will generally have a Select and an Edit push-button located nearby. The OEL Shell does not need to open any user files, however the user may wish to browse through the file hierarchy with the Motif File Selection Dialog.

To use the File Selection Dialog, choose the Select button near the file text that you want, and the File Selection Dialog will appear. The OK button will cause the selected file name to be copied to the text entry area in the control pane] that last had focus. You can focus on a text widget (move the mouse cursor over the text widget, and press the left mouse button) and then hit the OK button. Unlike other Motif programs written in the OEL, this File Selection Dialog is non-mortal. You may leave it up while you work with the main window. The OK button does not unmanage the dialog, so you can use it to fill filenames into several text widgets. The Cancel button will remove the dialog. The Help button will display help text for this dialog.

The Edit buttons will bring up an editor, which the user may choose in the resource file, to view the file specified by the contents of the currently selected text widget. The Exit button will cause the shell to terminate.

The shell also includes a Help button which is user configurable. This will pop up a single pane of help text. It is intended that

the designer of an OIL Shell also attach help to each widget in the work area. You may obtain help on any button or text field in the work area by selecting that object and then pressing the Help key. The default Help key for Motif application is the F1 key. To obtain

help on a push button without activating the button, move the mouse cursor off the button until the button no longer appears to be pressed. You may then release the mouse button without any activation.

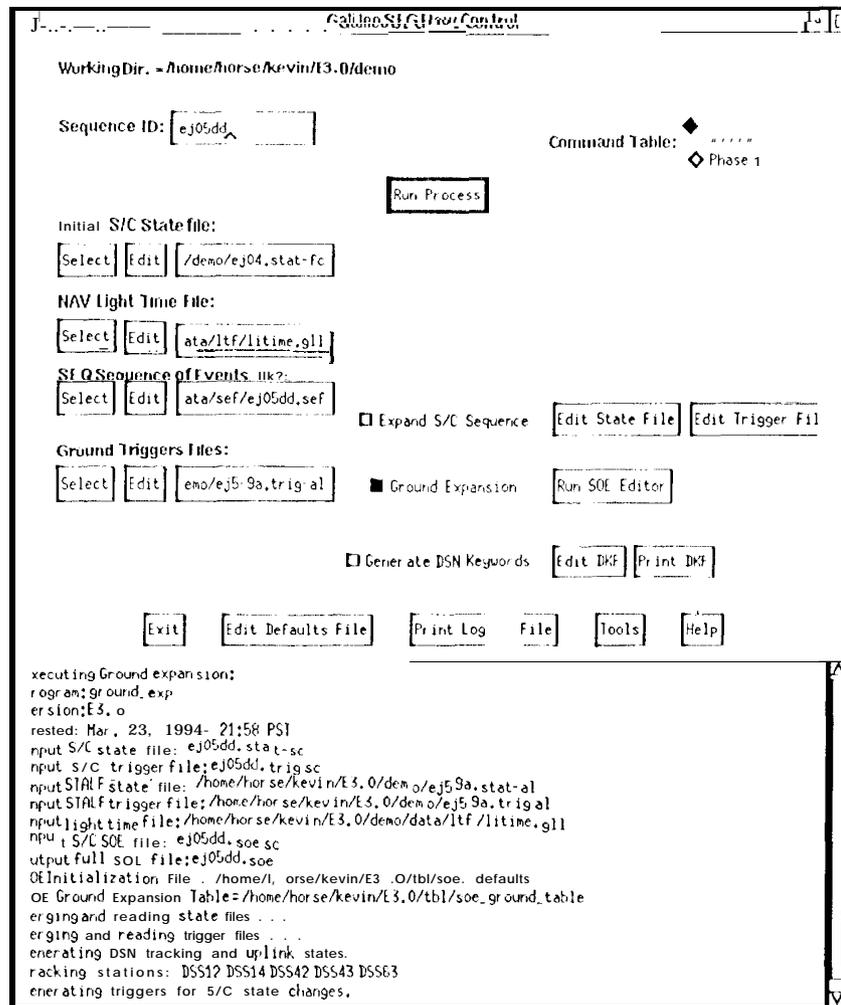


Figure 2. Galileo STALF Shell

WCL allows you to define the widget tree for an application in the resource file using new resource names such as `wcCreate` and `wcChildren`. In addition, callbacks are provided that set resources, manage and unmanage Widgets, run an external program and exit.

The OIL Shell is a very simple application built on WCL [2]. It is basically about ten callback procedures which may be used in the

resource file. The most important of these is `CmdCB` (the `command` callback). This callback executes its text string argument, for example, you could create a push button to execute the UNIX `ls` command as follows:

```
demo*!sPb.wcCreate: XmPushButton
demo*!sPb.labelString: Show Files
demo*!sPb.activateCallback: CmdCB(ls)
```

A simple command like this could be executed with WCL alone. The OI]. Shell permits one to access text widgets, toggle widgets, and option menus, and pass these in CmdCB. For example, if demoTog is a toggle button, then \$demoTog[-r] has the value in the brackets if the toggle button is true, and is the empty string if false. Likewise, the value of a text widget is just the text that the user entered.

Another very useful callback is the FocusCB which is used to specify the directory filter string used with the File Selection Box. A FocusCB is used with each text widget that is used to contain input file names.

Besides the resource file, two other files are used by the OI]. Shell. These are a drawing file, that places simple, X 11 primitives (not widgets) in the drawing area. This has been used to give the OI]. Shell the appearance of a flow chart. The other file is the help file.

Output from the child processes is sent to the scrolling message area below the work area. In addition, there are some special text messages that the child can send back which set resources in the OI]. Shell.

In addition to SIG, we have used the OI]. Shell for many other functions in operations. These include training, generating database queries, and running a command compiler.

Some advantages of using the OI]. Shell are:

- It is easily configurable by operations personnel.
- It separates the computing engine from the GUI, thus simplifying testing of the computing engine.
- All functions may be run with or without a GUI.

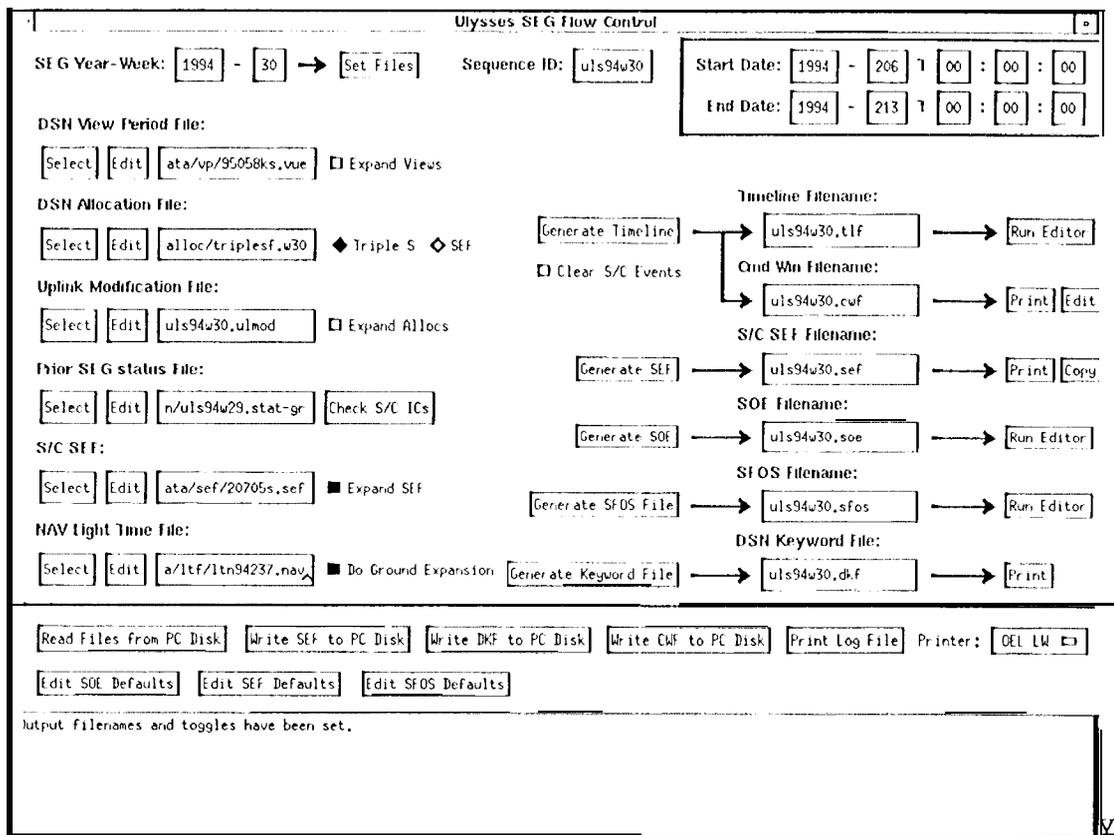


Figure 3. Ulysses SIG Shell

ULYSSES SEG

We have recently adapted the SEG software for Ulysses. The Ulysses mission is considerably different from the other missions in that the primary command mode is real time. Thus we do not have a spacecraft sequence file as an input to SEG. We introduced a new graphical document called the Timeline which contains the DSN allocation, view periods, and command windows translated into spacecraft time. This is generated from the DSN allocations and view periods files, and the lighttime file. The SEG operator then uses these times to schedule the spacecraft. Typical activities scheduled include: records and playbacks, telemetry mode Changes and maneuvers. Since the SIOS editor is a general purpose timeline editor, it is also used to edit the Timeline document. The spacecraft information is then extracted and put into a file that roughly corresponds to the spacecraft sequence file for other missions.

From this point on, Ulysses SIOS resembles SEG for the other JPL projects. The telemetry state of the spacecraft is extracted from the sequence file. The ground events are generated for the beginning and end of each track, the DSN configuration, spacecraft telemetry state changes, and other significant activities. This information is then used to create the SIOS, SIOS, and DSN keyword files.

Ulysses SEG was the first project where the SIOS editor was used to input data that would then be passed on to other processes. The SIOS editor has functioned well, and it was easy to extract data from the SIOS records.

ACKNOWLEDGMENTS

This work was done at the Jet Propulsion Laboratory, California Institute of Technology, under a contract from the National Aeronautics and Space Administration. We would like to acknowledge the work of the technical staff in the OBI, the JPL Mission Operations

Teams, and the Ulysses Spacecraft Team for their enthusiasm and support.

REFERENCES

1. The Widget Creation Library, David E. Smyth, September, 1991.
2. OBI, Shell Programmers' and Users' Guide, Kevin J. Miller, October, 1993.