MODEL: A SOFTWARE SUITE FOR DATA ACQUISITION

C.Boissat, W.Bozzoli, P.Burkimsher, R.Jones, J-P.Matheys, G.Mornacchi,
T.Nguyen, D.M.Sendall, P.Vande Vyvre, A.Vascotto, D.Weaver

CERN
1211 Geneva 23
Switzerland

MODEL is a new suite of modular data acquisition software. It is aimed at the needs of LEP experiments, and is also general enough to be more widely used. It can accomodate a variety of user styles. It runs on a set of loosely coupled processors, and makes use of the Remote Procedure Call technique. Implemented originally for the VAX family, some of its services have already been extended to other systems, including embedded microprocessors. The software modules available include facilities for data-flow management, a framework for monitoring programs, a window-oriented human interface, an error message utility, a process control utility and a run control scheme. It is already in use in a variety of experiments, and is still under development in the light of user experience.

CERN LIBRARIES, GENEVA

CM-P00059926

(Presented at the Computing in High Energy Physics conf.
Oxford, April 1989)

Introduction

The MODEL software has been developed by the Online Computing Group of the CERN Data Handling Division. It provides an environment for data-acquisition and related applications. This paper outlines its main features and components, and indicates current areas of development.

The aim of the MODEL project was to contribute to the online systems of experiments at the Large Electron Positron (LEP) collider at CERN. It was intended to be general enough to be applicable to a wide range of other experiments. MODEL offers a range of modules that can be integrated into the data-acquisition software of a large experiment. It complements and extends the services available from the operating system and other lower-level software. The user can select rather freely from the modules available. Furthermore, individual modules have as few built-in assumptions as possible, so that there are few constraints on the overall architecture. This implies that the responsibility of system integration is left to the user.

For smaller experiments, there is a need for a self-contained system which takes care of the essential functions of data-acquisition, and which can be simply tailored to the user's needs. Such a system can be built from a kernel of MODEL modules configured in a standard way. Users can then get started rapidly with a minimum of effort, and concentrate on writing monitoring programs and specifying the parts of the system which are experiment-dependent.

MODEL was originally intended to run under the VMS operating system on computers of the Digital Equipment Corporation VAX family. The typical environment is a set of VAXes, microVAXes or VAXstations linked by DECNET. The extension of MODEL services to embedded microprocessors is under way, and is already in use in several experiments. We are currently studying the porting of MODEL to operating systems other than VMS, with emphasis on the use of Open System Foundation standards.

The organisation of data-flow

Within a given VAX, event data must be distributed to recording media and monitoring programs. This is done by the **MODEL Buffer Manager (MBM)**, which organises the flow of events in **streams**. Events are fed into a stream by so-called **producers**, and can be taken out by **consumers**. A typical producer might be a program which reads events from front-end electronics. The most important types of consumer are monitoring programs and data recording programs. A **MODEL Standard Recorder (MSR)** handles recording on disk or tape in the CERN standard formats,

A stream can have more than one producer; the data from these are fed into the stream independently. Consumers are attached to **ports**; the user can define what type of events, and what fraction of them, are seen by consumers attached to a given port. If events are to be processed by user programs before being recorded or monitored, **stages** can be defined. A program can then read events, process them, and reinject them into the data stream. The user has a wide choice of how to set up streams, ports and stages; the routines which set them up can be called from any program.

The **MODEL Standard Consumer (MSC)** is a flexible data-dump program for events coming from the MODEL Buffer Manager (MBM). Since all the features of MBM can be exercised interactively, MSC can be used as a tool to understand the capabilities of MBM. It obtains the current MBM configuration automatically, and uses it to construct menus which enable the user to select a port and stage. MSC can be connected to any port, and can be switched from one port to another. It can also create new ports, with sampling characteristics defined via menus.

Monitoring programs

There are few constraints on the style of monitoring programs, or on the way they make use of the MODEL services. A **Monitoring Process Frame (MPF)** is available for use as a template. This provides a structured set of procedures making up a framework for monitoring processes. Users include their own procedures in this frame to perform the actual monitoring of the data, relying on MPF procedures for interfacing to the various MODEL packages. A typical monitoring process will have four main phases: initialization, operator interaction, data processing, and termination. Each of these is handled by a user-written procedure. Start and end of run procedures can also be included. If they wish, users can replace some of the standard MPF procedures with their own versions.

The user-written procedures influence the execution-flow of the process by means of return codes. External occurrences, such as an operator asking to communicate with the process, or MBM signaling the availability of an event, also influence the execution-flow of the process. MPF procedures detect these occurrences, and call the appropriate user procedures.

Interacting with the experiment

The **MODEL Human Interface (MHI)** is described in more detail in a separate paper [1]. It provides a framework for communication between processes running in the online system and people working at terminals or workstations. It offers a set of packages to help users to write interactive applications. MHI is window-oriented; applications are independent of whether windows are on a workstation or a terminal. It supports a distributed environment: a process can communicate with a window on any station or terminal in the multiprocessor system. This gives the user the freedom to migrate applications.

The basis of the system is a **Window Manager** which can be used either directly by application programs or via higher-level **Menu, Panel and Dialogue packages**. Applications written using these are insulated from the choice of the underlying hardware or software. Terminal emulation is supported for Fortran input/output. A GKS (ISO 7942) graphics service allows applications to use a graphics window on a remote machine. The KUIP [2] interface package is supported in command mode via terminal emulation, and an MHI style for KUIP is being implemented.

Communication between the process running the application and the process owning the terminal can make use either of Remote Procedure Call [3] or X-Window [4]. A pilot implementation of part of the system has also been made using DEC-Windows. These mechanisms offer independence from the operating system and the networking method used. They thus permit the extension of MHI services to systems other than VMS. An implementation for the OS9 operating system on the Motorola 68k microprocessor family allows applications running on these microprocessors to communicate with windows on a VAX system connected via Ethernet.

Error reporting

The **Error Message Utility (EMU)** provides a mechanism to route error messages at run-time. EMU implements a **pipeline**; messages are injected from user applications at one end and pass down the pipeline for processing. The four steps in error message processing are: Message Injection, Decoding, Routing, and Destination Processing.

The **Message Injection Routines** are called directly by application programs. They inject a message identifier, plus parameters if any, into the EMU pipeline. Messages are automatically sent to the **Decoder**, which retrieves the text for each message from a local **Message File**, and inserts the

values of any parameters into it. The Decoder also extracts from the message file any **properties** that are to be associated with the message; these are tag strings defined by the user. The decoded message, comprising text, parameters and properties, is next sent to the **Router**, which delivers them to one or more **destinations** according to instructions in a **Route Map** file. Routing decisions may depend on message properties.

Destinations may be physical devices such as a screen or disk file, or other programs. Programs are usually included at this point to format messages. Messages for some destinations, such as log files, will need no re-formatting. A simple scrolling screen formatting program is provided as a standard option, and users may implement destination modules for other devices. Destination programs need not be restricted to formatting; they may collect statistics, act as watchdogs, stop a run, or take action on receipt of an alarm. Message properties may be used to determine what action is appropriate.

The EMU pipeline may be re-configured by the insertion of additional user-written stages, for instance a message filter. The different stages of the pipeline need not all run on the same machine. Furthermore, several separate EMUs can be set up to co-operate. The output of a Router running on one machine can be connected to the input of a Router running on another. In this way messages can be displayed on more than one machine.

Control and organisation

Process control and distribution

The **MODEL Process Control (MPC)** system offers a set of services for distributed process control to enhance those provided by the VMS operating system; it extends these services to the distributed environment. Processes can be created, started, stopped, paused or resumed, and to have their status examined, by remote processes or users.

In cooperation with MHI, MPC provides the mechanisms whereby a process can interact with a user on one machine or device at one time and with a different user on some other machine or device later. To achieve this it maintains databases with information on processes, machines, users and terminals. It provides a simple menu-driven user interface so that users can make process control requests to remote processes, or establish communication with them. It also offers a uniform process termination signalling scheme.

Run control

Run control usually involves the execution and synchronisation of many different procedures. Requirements vary from experiment to experiment, and also within an experiment as setting-up proceeds and experience is gained. Similar problems are posed by calibration, or cold start of the apparatus. We have developed, jointly with the DELPHI Collaboration, an approach which can be applied to a wide range of such control applications. This is based on a **State Manager**, which incorporates a model of the different activities to be organised. The model is expressed in terms of **objects**, which are logical subsystems or areas of activity; examples might be data-taking or gas control. For each object a number of **states** can be defined: for instance the data-taking object might be in the active, paused, or stopped states. **Commands** can be sent to these objects, causing them to perform **actions** and to change state.

The user prepares a description of all objects, states, and actions. A compiler then uses this description to generate the State Manager for the experiment. The State Manager runs as a VMS process: it accesses the outside world by means of other processes provided by the user. The State Manager responds to external commands from an overall control process, as well as to

changes of state in associated processes. It then manages the resulting changes of state according to the pre-defined description. Associated processes communicate with the State Manager via a set of user-callable subroutines.

Integration with embedded microprocessors

The VALET-Plus is a VME-based microcomputer system incorporating a Motorola 68k-family processor and, in stand-alone mode, a link to a Macintosh, IBM-PC or other personal computer. It runs a simple data-acquisition system called SPIDER. The data-acquisition systems of some CERN experiments incorporate multiple VALET-Plus systems controlled from VAX or VAXstation hosts [5]. In such a configuration, the VAX must take over the functions of the personal computer used in stand-alone mode. The RPC system [3] has been adopted as the communication mechanism between front-end VALET-Pluses and host VAXes, and services have been implemented to control multiple VALET-Pluses from the VAX and integrate them into the MODEL environment.

SPIDER run control, event analysis, and debugging can all be performed on a VAX. A **MODEL SPIDER Producer (MSP)** running on the VAX presents menus and panels to allow the user to control the SPIDER systems. MSP acquires events from a SPIDER front-end and injects them into the MBM data-stream on the VAX. Since MSP uses most of the tools available for remote control of VALET-Plus systems, it can also be used as an example for users who wish to build their own applications.

The EMU injection routines can be called from the VALET-Plus environment. A process running on the VAX receives these calls and injects the messages into the EMU pipeline. All the facilities of the EMU error message system are therefore available to VALET-Plus applications, and are used by SPIDER and the other standard VALET-Plus software.

VALET-Pluses can be declared to the MODEL Process Control system on the host VAX. This permits VALET-Pluses to be controlled in a similar way to VMS processes. Commands intended for a VALET-Plus can be executed automatically from a DCL command procedure invoked by MPC.

A Valet State Manager process runs on the host VAX. At startup it reads the list of known VALET-Pluses from the MPC database, and maintains information on their states. It updates this information by making status requests to the VALET-Pluses over the RPC system. This information is available to any VMS process; a display process shows the current status of all the VALET-Pluses in the system.

Although effort has been concentrated on the integration of VALET-Plus systems, this approach could be extended to embedded processors running other software. Some experiments have set up MODEL-based systems communicating with processors running the OS9 operating system.

Conclusions

The MODEL software has already made a substantial contribution to the online systems of three of the four LEP experiments. It is also coming into widespread use in other experiments at CERN and elsewhere. The range of user styles adopted has been rather wide, and justifies the use of a modular adaptable system which allows experiments to establish their own conventions.

Acknowledgements

We wish to thank M.Berutti, L.Bessone, F.Cane, F.Gagliardi, G.Morpurgo and B.Wessels, who have contributed to the project at various times, and J.Barlow, B.Franek and M.Jonker who collaborated in the design of the SMI system. We should also like to thank our first users, who have helped to improve the specifications and test the system.

References

- (1) C.Boissat et al, *The MODEL Human Interface; Proceedings of the Conference on Computing in High-Energy Physics, Oxford 1989 (North Holland, 1989); CERN – DD/89/11.*
- (2) R.Brun et al. *PAW, a general-purpose software tool for data analysis and presentation: Proceedings of the Conference on Computing in High-Energy Physics, Oxford 1989 (North Holland, 1989); CERN – DD/89/17.*
- (3) T.J.Berners-Lee, *Programming Distributed Systems; Remote Procedure Call: Proceedings of the ESONE Conference on VMEbus in Research (North Holland, 1988) p 159.*
- (4) D.Myers, *HEP Graphics; Standards and portability versus performance and cost: Proceedings of the Conference on Computing in High-Energy Physics, Oxford 1989 (North Holland, 1989); CERN – DD/89/8.*
- (5) Y.Perrin et al, *The VALET-Plus Embedded in Large Physics Experiments, Proceedings of the ESONE Conference on VMEbus in Research, (North Holland, 1988) p 59; CERN – DD/88/19.*