Automatic Generation of Configuration Files for a Distributed Control System

J.Cuperus, A.Gagnaire

PS/CO Group, CERN, CH-1211, Geneva23, Switzerland

ABSTRACT

The CERN PS accelerator complex is composed of 9 interlinked accelerators for production and acceleration of various kinds of particles. The hardware is controlled through CAMAC, VME, G64, and GPIB modules, which in turn are controlled by more than 100 microprocessors in VME crates. To produce startup files for all these microprocessors, with the correct drivers, programs and parameters in each of them, is quite a challenge. The problem is solved by generating the startup files automatically from the description of the control system in a relational database. The generation process detects inconsistencies and incomplete information. Included in the startup files are data which are formally comments, but can be interpreted for run-time checking of interface modules and program activity.

INTRODUCTION

The block diagram of the CERN/PS accelerator control system is represented in fig 1. The Control Modules [1] are software modules which present a uniform call interface to the users. These Control Modules address the Equipment Interface, to which the accelerator equipment is connected. The Equipment Interface is composed of a software part: programs and interface module drivers, and a hardware part: crates with plug-in modules. The Equipment Interface is controlled by Device Stub Controllers (DSCs), which are microprocessors sitting on a VME board.

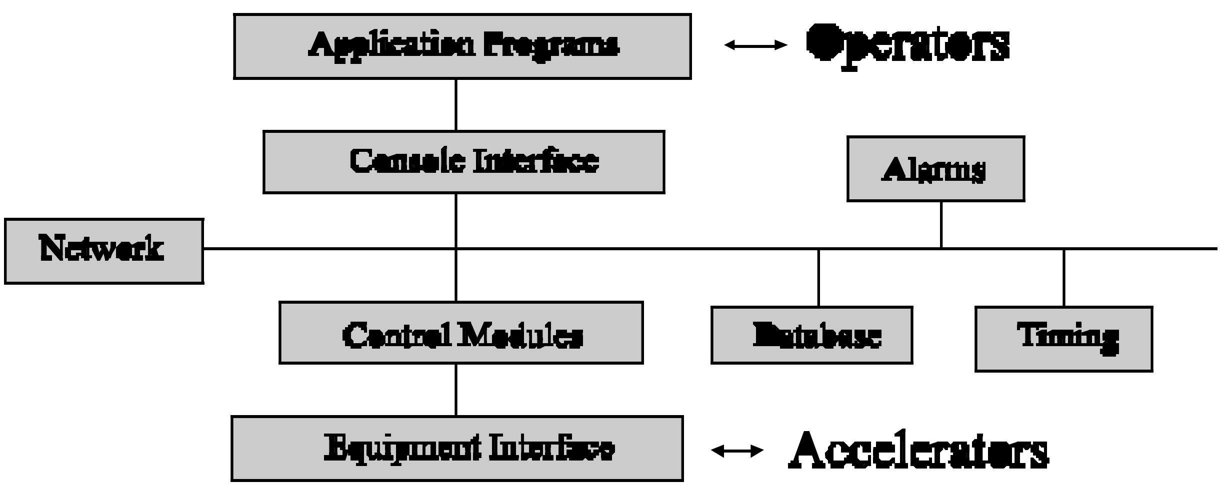

fig 1: Block diagram of the control system. What interests us here is the Equipment Interface, which is composed of programs, drivers, and hardware modules in VME, CAMAC, G64, and GPIB crates.

The Problem

The LynxOS operation system (a real-time UNIX) expects to find a rc.local initialization file for the DSC, with information about programs to be started, drivers to be installed, addresses, and interrupt vectors. Initialization files are typically 100 lines long and filling them in by hand is quite a problem:

- We have now more than 100 DSCs and the management by hand of each rc.local file is quite difficult and a waste of time.

- The startup sequence is made of related pieces and any change can have perverse side effects and lead to a fatal system fault or misbehavior of application programs.
- There is no validity check for conflicts in address ranges or interrupt vectors and the declarations are not checked against the really installed hardware.

The Solution

In fact, we had most of the data already in our relational database and, with a few extensions, we can now get all data from tables in the database. Filling in these data, with the help of forms, is easy and the tables are organized in such a way that duplication of information is avoided . Most parameters have default values and must be filled in only when a different value is required, which is exceptional.

We decided to use this information to generate the rc.local files automatically with a data driven program. This program performs several checks on the data and informs the user in case of conflicts. We will now look at this process in some more detail.

HARDWARE DESCRIPTION

Accelerator Interface Configuration

The hardware interface is connected to the rest of the control system through a DSC, which is a front-end microcomputer sitting on a VME board. The CPU is a Motorola MVME167 microprocessor. On one side it is connected to the Ethernet network and on the other side to the VME bus. All hardware is controlled through the DSC, either directly though modules (cards) on the VME bus or, indirectly, through VME driver modules for CAMAC, MIL1553, or HPIB loops (fig. 2).

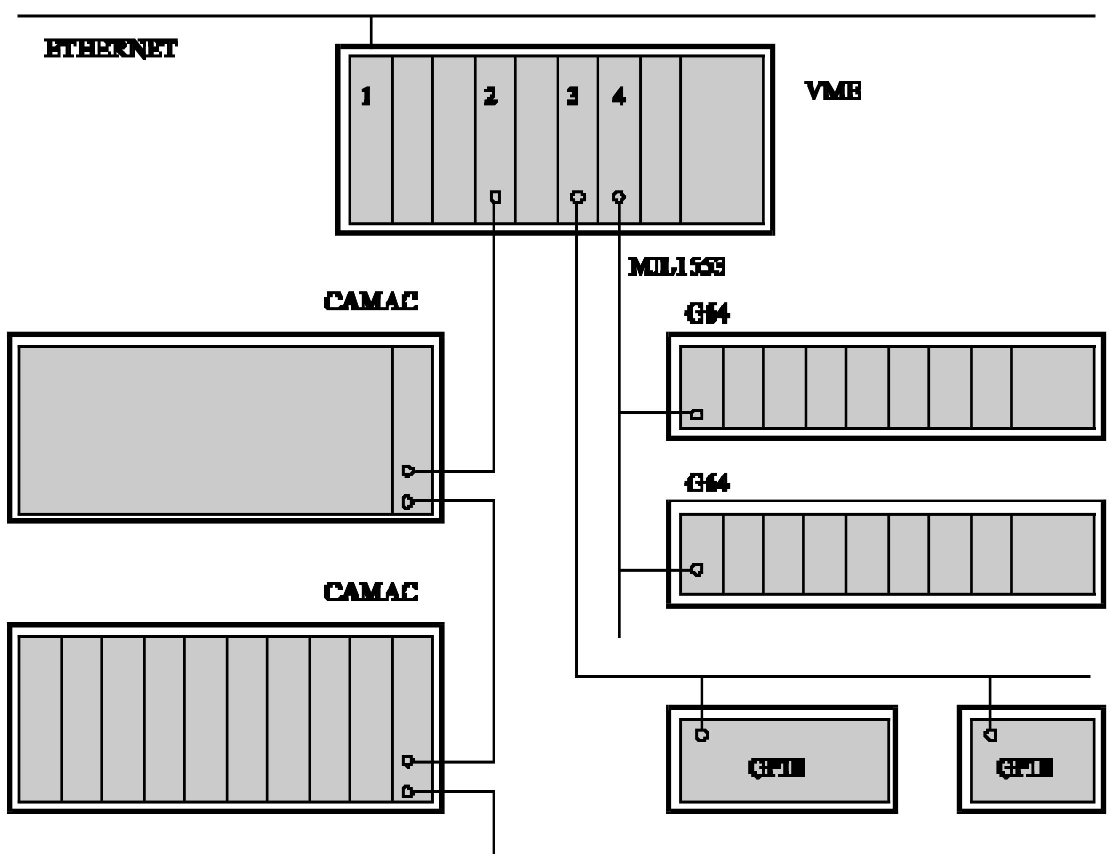

fig. 2: Accelerator interface configuration: (1) DSC microcomputer, (2) Serial CAMAC driver, (3) General Purpose Instrument Bus driver, (4) MIL1553 driver.

On these loops (or busses) can sit CAMAC, G64, or GPIB crates with slots for modules. These modules are, in general, connected to the accelerator hardware: power-supplies, instruments, actuators, etc...

Database Tables describing the Hardware

The whole interface configuration is described in tables in the Oracle relational database management system. We can distinguish two groups of tables:

- *TYPE* description tables, which contain a description of the object type or class. Tables COMPTYPES, CRATETYPES, and MODULETYPES contain all the fixed attributes per type of computer, crate or module.
- *INSTANCE* tables which describe the implementation of an instance of the type. The installed computers, crates, and modules are instances of types but they have attributes of their own, which are entered in following tables:

COMPUTERS = { CompName + CompType + . . . }
CRATES = { Crate_Id + CrateType + CompName + LoopNo + CrateNo + . . . }
MODULES = { Mod_Id + ModuleType + Crate_Id + Slotno + Seqno + . . . }

The type of computer that interests us here, is the DSC. A crate is any chassis with slots of type VME, CAMAC, G64, or GPIB. A module sits in a slot of a crate. Additional details about module instances can be found in following tables:

MOD_EXCEPTIONS = { Mod_Id + DriverName + InterruptLevel + BaseAddress1 + . . . }
MOD_INTERRUPTS = { Mod_Id + InterruptNo + Subaddress }

The MOD_EXCEPTIONS table is separate from the MODULES table because modules need an entry only when the values calculated by default are not suitable, which is quite exceptional. The MOD_INTERRUPTS table is separate from table MODULES because a module can generate several interrupts.

Finally, there are the tables EQUIPMENT and INSTVAL, which are related to the Control Modules and which contain, among other things, physical addresses for VME, CAMAC, G64, or GPIB hardware modules, for controlling pieces of accelerator hardware.

SOFTWARE DESCRIPTION

Software Modules

Software modules come essentially in two kinds:

- Hardware drivers which hide some of the intricacies of the hardware control for a type of hardware module.
- Programs which provide some control or surveillance function.

Some of the external aspects of software modules can be described in Oracle tables.

Database Tables for describing the Software

We can again make a distinction between type description tables and instance description tables. The different driver types are described in following table, with default parameters and expected tags for startup sequences:

DRIVERTYPES = { DriverName + DriversPrio + Address_Tag + . . . }
DSCPROGDEFS = { StatupName + Progname1 + Progname2 + StartSequence + . . . }

The StartSequence can contain placeholders \$1..\$4, which can be replaced by parameters in table DSCPROGRAMS. This table contains the list of programs to be started in each computer, and drivers to be installed:

DSCPROGRAMS = { CompName + Seqno + ProgName + Prio + Params + . . . }

DATA ENTRY WITH FORMS

All data are entered into the database through Forms, of which fig. 3 gives an example. All forms for this application, together with provisions for starting all necessary programs and actions are grouped in a menu structure. This menu structure is the only interface needed for entering data and generating the needed configuration files. After filling in the data and choosing a DSC from a list, you can select the “generate configuration file” item from the menu.

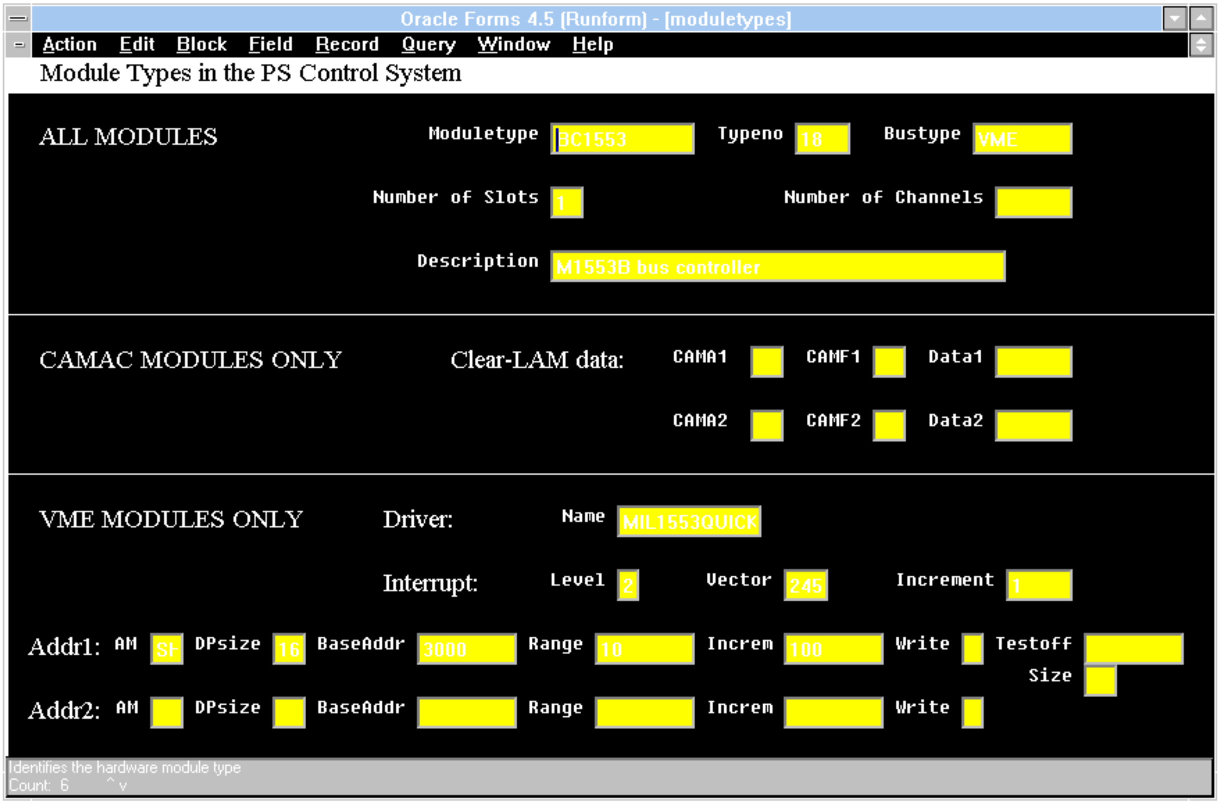

fig. 3 A data entry form

FILE GENERATION

Default Address and Interrupt Vector Calculation

Every VME module type has one or two base addresses and an interrupt vector. The first module of this type installed in a crate (with Logical Unit Number - or LUN - equal to 0) adopts these values as defaults. Subsequent modules of the same type installed in this crate get LUNs 1, 2 , 3 ... and the following default values:

ADDRESS = BASEADDRESS + LUN*ADDRESS_INCREMENT

INTERRUPT_VECTOR = BASE_VECTOR + LUN*VECTOR_INCREMENT

If none of the address ranges or interrupt vectors in the crate overlap, then all is OK. If not, different values must be entered in the table MOD_EXCEPTIONS. The need for this is indeed exceptional, if the type defaults have been selected with care.

Configuration File Generation

The generation program asks for the name of the DSC and then, with the help of the data in the database, executes a number of checks:

- **Check VME, CAMAC, G64, and GPIB:** all equipment addresses must correspond to an installed module.
- **Check crate slots:** physical space must be available in the crates for all modules.
- **Check interrupts:** calculate interrupt vectors and check for conflicts.
- **Check base addresses:** calculate base addresses and check for overlapping ranges.
- **Check Drivers:** required drivers must exist and work within their limitations.

If all is OK, then the program generates the following entries in a rc.local file:

- **Write IOCONFIG data:** comments which describe the VME modules and crates in the interface. These are not required by LynxOS but are readable by a configuration display program. They are also used, at startup, by a program which checks that all listed hardware is installed and addressable.
- **Write first set of program startup data:** programs which must be started before the drivers.
- **Write driver installation data:** install all drivers, in correct sequence.

- **Write second set of program startup data:** programs which must be started after the driver.
- **Write CLIC data:** comments readable by a program execution monitor, named CLIC which alerts the operator when one of the programs stops functioning.

An Example of a rc.local File

```
#!/client/dlsh
#1995-SEP-29
# dmcrsync startup file rc.local, generated 1995-SEP-29/11:14.
$#set(path, . /client /dsc/local/bin /dsc/bin/bin /dsc/bin/rt /bin )

#****************************************************************************
#   WARNING :  File generated from database.
#              Can be overwritten at any time !
#
#   Latest dsc modifications made by : nmn_svps13_19Apr95_12:23
#****************************************************************************

# ***** IOCONFIG Information *****

#      ln  mln  module-type      lu  W   AM   DPsz  basaddr1  range1  W   AM   DPsz  basaddr2
    range2   testoff
##+# 1  0    VME SAC          0   N   SH   DP16  0         20      N   ST   DP16  0
    80000  -
##+# 2  0    VME SDVME        0   N   SH   DP16  f800      400     N   --   ----  0
    0      -
##+# 3  0    VME SDVME        1   N   SH   DP16  f000      400     N   --   ----  0
    0      -
##+# 4  0    VME SDVME        2   N   SH   DP16  b800      400     N   --   ----  0     0
    -
##+# 5  0    VME MVME147S     0   N   --   DP16  0         0       N   --   ----  0     0
    -
##+# 6  0    VME PLS-REC-FPI  0   N   SH   DP16  e000      1000    N   --   ----  0     0
    -
##+# 7  0    VME PLS-REC-FPI  1   N   SH   DP16  d000      1000    N   --   ----  0     0
    -
##+# 8  0    VME PLS-REC-FPI  2   N   SH   DP16  c000      1000    N   --   ----  0     0
    -
##+# 9  0    VME ICV196       0   Y   ST   DP16  500000    100     N   --   ----  0     0
    -

#  ln  sln   module-type      lu  evno  subaddr  A1  F1  D1  A2  F2  D2

#      ln  mln  module-type      lp  cr
##+# 10  2    CAM SCC-L2       1   11
##+# 11  2    CAM SCC-L2       1   12
.
.
##+# 24  4    CAM SCC-L2       3   8
##+# 25  4    CAM SCC-L2       3   12

# ***** Program Startup before drivers *****

# ***** Driver Initialisation *****

-( cd /dsc/bin/drivers/sacvme;  sacvmeinstall \
 -R0 -M0 -V254 -L2   )-

-( cd /dsc/bin/drivers/sdvme;  sdvme_v2install \
 -Af800 -V160 -L2   -LP1   -, \
 -Af000 -V161 -L2   -LP2   -, \
 -Ab800 -V162 -L2   -LP3   )-

-( cd /dsc/bin/drivers/fpiplsvme;  fpiplsvmeinstall \
 -BOe000 -BV172 -BL2    \
 -AOd000 -AV173 -AL2    \
 -COc000 -CV174 -CL2    )-

-( cd /dsc/bin/drivers/icv196vme;  icv196vmeinstall \
 -AO500000 -AV130 -AL2    )-

# ***** Program Startup after drivers *****

# Install data used by ioconfig library
-(cd /dsc/bin/drivers/ioconfig/; ioconfigInstall  )-

# Start errlogd reporting errors to mcrsrv
prio 16   errlogd mcrsrv </dev/null &

# Report messages from CLIC or IOCONFIGDIAG to mcrsrv
rm -f /dev/sysReport.pipe
prio 17 sysReporter mcrsrv &

# Survey of VME and CAMAC loops according to configuration
prio  10 /dsc/bin/bin/ioconfigDiag  </dev/null &
```

```
echo Restoring datatable and run Nodal sys_go
dtrest
if $access(r,all,/dsc/bin/rt/sys_go.nod)
nodal ``$se ex.fl="";lo /dsc/bin/rt/sys_go.nod;run"</dev/null
end

prio 21    global_event_server \
   /dsc/local/bin/global_event_configuration </dev/null &

echo start NODAL EXEC server
prio 19   usr_cmd mcrop `nodal  -s EXEC'

echo start NODAL IMEX server
prio 19   usr_cmd mcrop `nodal  -s IMEX'

echo start equipment RPC server
prio 19   server </dev/null &

# echo start CLIC alarm survey
# $#setenv(USER,root)
# rm -f /dsc/local/clic/.clic.lockdaemon
# rm -f /dsc/local/clic/.clic.lockrun
# prio 10 clic -s

# ***** Programs in Clic Survey *****

#-# errlogd
#-# server

# End of file rc.local : all OK.
```

CONCLUSIONS

The system is a great help in configuring our many DSCs. Instead of laboriously filling in the configuration files by hand, which is a very error prone process, we derive them from descriptions in the database. The database is easier to fill and minimizes the entry of repetitive information and the data are necessary for other purposes anyway. As a bonus, the hardware configuration is checked at startup and programs are surveyed at runtime. The descriptions are broad enough to cover all special cases and we never find it necessary to edit the configuration files by hand.

REFERENCES

1. L.Casalegno, J.Cuperus and C.H.Sicard, Int. Conf. on Accelerator and large Exp. Physics Control Systems, Vancouver, 1989, D.P. Gurd, M.Crowley-Milling, eds. (North-Holland, Amsterdam) p 412.

2. More information about the PS accelerator control system can be found on the web address: https://psas01.cern.ch/