MASSACHUSETTS INSTITUTE OF TECHNOLOGY
HAYSTACK OBSERVATORY
WESTFORD, MASSACHUSETTS 01886

22 October, 1996

Telephone: 508-692-4764
Fax: 617-981-0590

To: Mark 4 Software Group
From: Roger Cappallo
Subject: Conductor (rev. 2)

This document, and its subsequent revisions, will define the function and implementation of conductor.

PREAMBLE

Conductor is at the center (at least topologically) of the Mark 4 real-time software suite (see Figure 1). Its purpose is to orchestrate the variety of tasks that must take place within the real-time system in disparate modules, in such a fashion as to allow robust and flexible operations. The design proposed here is quite modular, so that additional functionality can be added to the basic nucleus, as the rest of the software system becomes better defined.

PURPOSE

Conductor will handle the sequencing of correlator operations, both for normal operations and for unexpected events, such as operator intervention or fault conditions. In the Mark 3A correlator system, the analogous sequencing operations were spread across several programs, including COREL, VRUN, VSYNC VJOKY, and SKD71. Operations which are not primarily sequencing in nature, such as creating/reading files, or interacting with hardware, will be handled by a set of discrete manager programs.

INTERACTING OBJECTS

Conductor's sole interface to all other objects in the real-time system will be via exchange of messages; it will neither read nor write files. The following modules/objects will interact with conductor:

- SUM - SU Manager: communicates with all of the SU's. Manages tape mounts/dismounts, SU data files (type 3 files contain phase cal & state counts), various error conditions involving the tapes (head positioning problems, missing data on tapes, …), scan pipelining.
- TAL - Tape Librarian: manages requests for information from the online tape library database;
- GEN - genaroot: calculates a priori delays and rates via CALC engine; fits quintic spline polynomials to tabular points; creates root files with delay and phase polynomials for each station
- OPI - operator interface: handles all operator interactions, including initiation of single scans, status requests, and diagnostic information (fringes, etc.)

- BFM - Batch File Manager: (merge with OPI??) generates correlation scripts for multiple scan processing

- CRM - CU Resource Manager: manages correlator and input boards; responsible for allocation, configuration, and status monitoring.

- CDM - CU Dataflow Manager: manages correlator output files; creates them and writes correlator lag and status data to them.

- RCM - ROT Clock Manager: sends realtime messages to SU's setting their ROT clocks, and thus controlling the detailed timing (pacing) of the correlations

- CLM - Correlator Log Manager: receives log messages from other modules, time-stamps them, and writes them to a log file associated with current scan

ARCHITECTURE

Conductor will be implemented as an interpreter for a table-driven state machine. It is unclear how all of the desired functionality could be achieved by merely stepping through states and passing messages. Such an approach was considered and abandoned, as it would lead to a proliferation of external message processors. Thus it is proposed that conductor also have internal actions associated with each state transition. These actions will be implemented as a small piece of C code which will perform the requisite activity. All of the large scale sequencing will be inherent in the state machine. The conductor code to implement actions will be restricted to local, immediate actions that require no waiting. Typical actions would involve such things as reading and writing to conductor's own data structures, making arithmetic and logical calculations, and deciding to post messages.

STATE TABLE

An FSM table entry would include four fields: <state>, <event>, <action>, <next state>. The FSM goes from <state> to <next state> when an <event> occurs, which in conductor is always the receipt of a message. The <action> which is taken might involve manipulation of data structures within conductor and the transmission of messages. Additional flexibility can be afforded by allowing the action routines to place messages upon conductor's input queue. One effect of this is to allow branching (i.e. state sequencing) decisions to be made in the routines, since they can generate an event to force a transition into the desired target state. The special state **any_state**, which will be matched by any state, is used as a way of handling general conditions not tied to any particular state. In such a case, the resultant state is not generally altered. The state machine searches the state table in order, trying to find a match for both <state> and <event>. The global events which have **any_state** as their <state>, would naturally be placed at the bottom of the table. Similarly, there is a global event matcher called **any_event** A skeleton FSM table can be found in Appendix A, while Appendix B contains pseudo-code for the actions indicated by the FSM.

MESSAGES

The messaging sub-system will be taken, as much as possible, from the EVN correlator. Their message format apparently has all of the capabilities we need, such as a message ID, reference ID, message source and destination, a time stamp, and variable format data blocks. Their system also allows messages to be queued for later execution at a specified ROT (we may find it useful to expand this capacity to allow a COT epoch for command execution). For more detail concerning the messaging system consult the EVN Memo #42. Appendix C contains a complete list of the messages that are sent and received by conductor.

MULTIPLE CORRELATIONS

One constraint which drives the design is the necessity to handle multiple concurrent correlations. These simultaneous scans can be handled neatly through use of multiple state pointers, in some ways analogous to a saved PC (program counter) in a multi-tasking system, and some ancillary state information, which would be analogous to saved registers. We can then define a state structure, which contains the current state pointer, and all of the state information needed to contextualize a scan (see Figure 2). When an event occurs, the software must figure out which scan it references, and switch context by loading the correct state pointer and state information, before operating upon the event. At the highest level, the program can be considered to be a state-processor, with the following sequencing:

1. read next event in queue
2. determine which state structure is in context, *I. e.* which scan is being referenced, from the message contents
3. load the state
4. process the event
5. save the new state
6. wait for next event in queue

PIPELINED SCANS

A design goal of the Mark 4 software is that scans will be processed as expeditiously as possible. Thus it is desirable to have scans be processed “on the fly”, or in a pipelined fashion whenever feasible, with continuous tape motion. In most experiments there will only be a 5 to 10 second gap on the tape between scans, caused by the tape stopping and starting at record time. As a result, the realtime correlator software needs to be flexible and responsive enough to be ready to start correlating the subsequent scan in a few seconds.

It is proposed that the basic responsibility for detecting scans which can be pipelined, should be given to the SUM, which will have access to the tape library database. The SUM, when tapes are requested, will need to examine the database of what is actually on the tapes, and see if the requested scan follows the current scan, on all tapes, and in the same direction. If so, it develops a useable time based upon its *a priori* information, and doesn’t actually try to manipulate the tapes. The intended pipelining is reported back to the conductor through the **valid time** message.

Appendix A - Skeleton FSM Table

STATE	EVENT (originator)	ACTION	NEXT STATE
await_root	**root_complete** (GEN)	**acquire_tapes**	**await_valid_times**
await_valid_times	**valid_time** (SUM)	**process_valid_time**	**await_valid_times**
await_valid_times	**have_all_times** (CON)	**make_cu_request**	**configuring_corr**
await_valid_times	**cant_find_time** (SUM)	**no_time_oper_alert**	**await_valid_times**
await_valid_times	**remove_station** (OPI)	**deactivate_station**	**await_valid_times**
await_valid_times	**time_status_change** (OPI)	**check_time_status**	**await_valid_times**
configuring_corr	**corr_resource_avail** (CRM)	**begin_correlator_pass**	**tape_startup**
tape_startup	**drive_synchronized** (SUM)	**add_synced_drive**	**tape_startup**
tape_startup	**all_synced** (CON)	**post_all_synced**	**correlating**
correlating	**end_of_scan** (CDM)	**end_correlation**	**<null>**
any_state	**correlate_scan** (OPI)	**setup_new_scan**	**await_root**

Appendix B - Action Pseudo-code

acquire_tapes:

save root filename, etc.

send message to SUM requesting tape mount/access

process_valid_time:

store time in **time_array[station]**

change **station** status to **time_found**

if all stations have **time_found** status **assert_event** (**have_all_times**)

make_cu_request:

send message to CRM requesting necessary correlator resources

begin_correlator_pass:

determine initial ROT clock setting

send message to RCM to start at the initial time ASAP

no_time_oper_alert:

send message to OPI reporting missing time

deactivate_station:

send message to SUM removing designated **station**

assert_event (**time_status_change**)

setup_new_scan:

initialize new scan thread

send message to GEN requesting root generation

add_synced_drive:

update current state to reflect drive status

if no more drives unsynced **assert_event (all_synced)**

end_correlation:

send message to CLM logging scan stats

remove current scan's state structure from list

search other threads for applicable pipeline-pending; act upon result

Appendix C - Conductor Messages

message	source	destination	data	comments
generate_root	CON	GEN	ccf name	
			scan time	
corr_resource_req	CON	CUR	resource specifier	
			wait bit	set if OK to wait
request_tapes	CON	SUM	station list	
			scan time	
have_all_times	CON	CON		
configure_tapes	CON	SUM	track configurations	
define_outfile	CON	CDM	filename structure	
start_rot_clock	CON	RCM	ROT	
current status	CON	OPI	processing state	
root_complete	GEN	CON	root filename	
valid_time	SUM	CON	station	
			first useable time	
			pipeline status	
cant_find_time	SUM	CON	station	
CU_resource_avail	CRM	CON	track configurations	
remove_station	OPI	CON	station	
correlate_scan	OPI	CON	ccf name	
			scan time	
			station list	
query_status	OPI	CON	scan time	