VLBA Correlator Memo No. 98

Correlator Software Status*

Don Wells
National Radio Astronomy Observatory, Charlottesville, Virginia

July 17, 1990

Abstract

A status report for the real-time software project for the VLBA Correlator for the period of approximately 2Q90 is given. During this period the conventions for code management and style were settled and portions of the tasking structure came into existence. The Group has nearly completed bringing all existing code and code development under the NSE code management system; i.e., most of the Group members are now working together on a shared body of code rather than working independently on separate subsystems.

Contents

*Postcript file in: /home/ccc/vlb/fxcorr/dwells/station1/doc/jul90.ps

1 Naming & Documentation Conventions

Until recently Group members were working independently on subsystems of the Correlator, and coding conventions differed. As the code is being integrated and members are beginning to share responsibility for it, the time has come to more closely coordinate styles and practices, while continuing to tolerate some diversity.

1.1 Use of WRS's Module Naming Conventions

The Group decided during 2Q to adopt the WRS [Wind River Systems] convention of naming all modules with a 3- or 4-character lower case classification prefix followed by one or more capitalized (*not* uppercase) words. For example, all module names in the `:vxWorks:station` component now start with "`stn`". This naming convention also applies to our include files For example, the include associated with the station component is `<stn.h>`. External structure declarations (for globally known variables, the C analogy to Fortran `COMMON` blocks) have names like `stnExtern`. Finally, the convention is even used for parameters defined in includes and for enumeration symbols. There are two motivations for the convention:

- When a programmer finds a symbol used in any module he can immediately predict in which include file or NSE component (directory) the symbol is defined.
- The convention minimizes the possibilities for name conflicts in the shared memory, shared library global namespace of vxWorks.

1.2 Commenting/EXDOC Conventions for C-code

The Group has reaffirmed our prior decision to use module commenting conventions similar to those used by the Monitor & Control Group and to use a variation on the program `EXDOC` to extract the comment text, keywords, one-line descriptions, etc. In particular, the Group has reaffirmed that the one-line description summaries and keywords are a good thing, and will provide a toolset to extract, search and print these items. These decisions have not yet been fully implemented.

2 Tasks

During the period covered in this status report the tasking structure for the Correlator began to take shape. In particular, code for tasks associated with the station, model-generation and tape-control functions was defined, and these functions are discussed below.

Development of the batch jobs associated with the DBMS on the general purpose computers was suspended beginning 2Q90 to permit concentration of the whole Group on the RT tasks. It was judged that this DBMS code was in good shape, with development substantially ahead of the RT code. Further development of the DBMS schema is now limited to what is found to be needed for further RT task development. This hold on DBMS development will probably persist through 3Q90 and maybe through 4Q as well.

All development of the Archive, Clock and Distribution tasks has been deferred temporarily. Also, the procurement of hardware for the Archive and Distribution functions has been deferred until sometime in 91 (maybe 2Q91) both because the software development is deferred and in order to exploit continuing technology developments.

2.1 Station Code

The Group decided at the beginning of 2Q that we would concentrate on integrating tasks along an axis connecting the job script files which are produced by the DBMS and the HCB [Hardware Control Bus] which connects to the FFT cards. (This axis is the vertical line of arrows in Figure 1 below.) Subsequently we broadened the axis to include the model and tape tasks. The goal is to deliver a first version of the integrated task structure which will run jobs, commanding the hardware as it becomes available, but also able to run independent of the hardware for software debugging purposes. The `stnTask` interacts with most of the hardware and software entities in the architecture, and so the act of defining it necessarily defines most of the rest. Indeed, paradoxically, `stnTask` cannot be defined until the other entities have been defined; development of `stnTask` is therefore lagging behind development of several other related tasks.

In Figure 1 the tasknames follow our new naming conventions. This figure is an updated version of Fig. 4 on p. 36 in VLBA Correlator Memo No. 95 (Sept. 29, 1989); it is beginning to be an "as built" schematic. The current experimental version of the system spawns the "permanent" tasks `tickTask`,

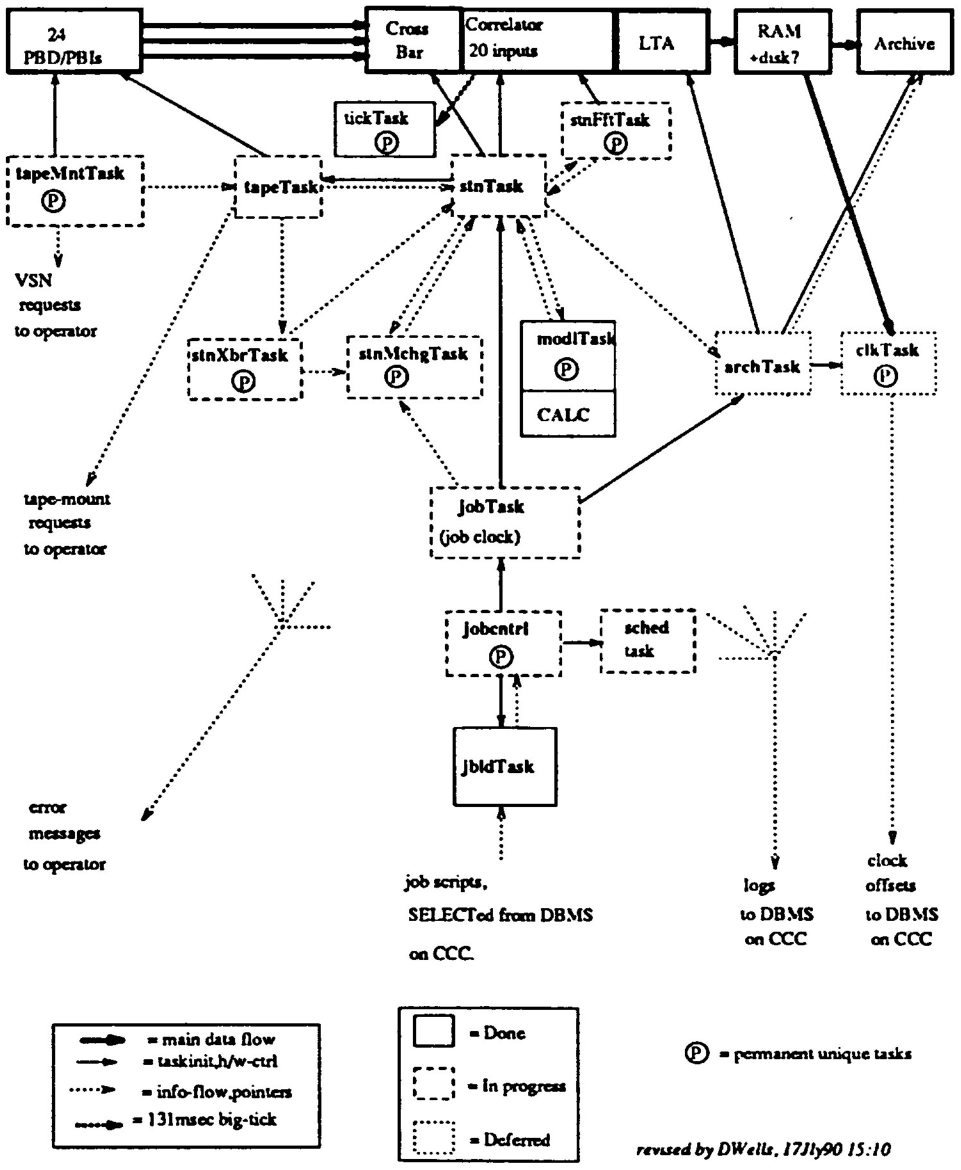

Figure 1: Real-Time Tasks

tickTest, **stnFftTask**, **modlTask** and **stnMchgTask**. These tasks are specially marked in the figure as "permanent unique"; the latter attribute refers to the fact that there is only one instance of each of these tasks, the former attribute refers to the fact that these tasks are spawned at boot time and run forever. The experimental code also runs the **jbldTask** to read a test job script, and then spawns a **jobTask** to run the job, which then spawns one or more **stnTasks**. A version of the experimental code in June spawned two independent job tasks with different job scripts, each of which spawned multiple independent station tasks, just to demonstrate that the architecture supports this. The **jobTasks** and **stnTasks** are created dynamically and go away when their respective jobs are completed. Soon the **stnTasks** will also be spawning **tapeTasks** and eventually the **jobTasks** will be spawning **arryTasks** and/or **archTasks** (these details have not yet been finally decided).

Because the hardware interrupt has not yet been connected, and also for test purposes, the 7.63 Hz ($10^6/2^{17}$) "ticking" action of the VLBA Correlator is simulated with a software interrupt generated by the **tickTest** task that runs at a lower frequency than the RT hardware will run. Each **jobTask** maintains the master time variable for its job. We decided that we will use floating point MJD (Modified Julian Days) for our internal time scales.

2.2 Model Code

During 2Q90 the **modlTask**, which includes the GSFC CALC code, approached its final form. We decided to compute models on a station basis. The model task acts as a server and each station invokes it periodically by writing messages into a "pipe". The model task reads the messages and then examines the station's job script and computes models as needed, and appends them to a list of models maintained for each station. The objective is to compute models far in advance of need, about 20 minutes ahead in the current version, but not for entire jobs. Thus, the work of computing models, which is expected to be the major CPU loading in the first CPU, is spread out and scheduled to minimize risk of a hard deadline crisis. The station tasks get their models from the lists as needed and delete them from the lists when they are no longer needed. WRS's **lstLib** has proven to be useful for these operations.

We have added the **stnMchgTask** (model changer) to the software architecture in order to coordinate the simultaneous loading of new delay models. Because there is only one time variable for the model generator system of the FFT control card, and because this variable resets to zero on a model change, the act of loading or changing any model is a critical region for the entire Correlator, analogous to a crossbar reconfiguration. Indeed, crossbar changes and model changes must be interlocked, and furthermore any crossbar change must be followed immediately by a model change.

2.3 Tape Code

An experimental standalone tape manipulation code has been developed during 2Q, code that talks using the MCB to our one playback drive. A goal for 3Q is to integrate this code into the experimental tasking structure. This experimental tape code also incorporates our first use of the "screens" package for operator interfacing.

3 Use of NSE

The decision to utilize Sun's CASE product "NSE" [Network Software Environment] for the Correlator project was not justified by any prior experience in the astronomy community. Indeed, there is little useful experience with such products in the larger general computing community. In this case the VLBA Correlator project is testing new technology. We believed that the gain would prove to be worth the pain over the lifetime of the project. Although the jury is still out on this trial, the counsel for the defense is beginning to be fairly optimistic. We have indeed suffered from the "learning curve" of an unfamiliar, very sophisticated, software product, and we have suffered from certain outright bugs. But already the ability to coordinate parallel development is beginning to pay off in the expected fashion.

3.1 Component and Environment Structure

During 2Q almost all members of the Group began to use the component and environment structure for their code management. The component structure as it stood on 15-July-90 in the author's "`station1@dwells`" environment is shown in Table 1; this table was adapted from output produced by "`nsecomp list -r :`".

The intent of the hierarchical component structure is to decompose the software into logical functional modules and categories. This is both to facilitate the automated delivery of modules of vastly different type and also to make it easier to find particular modules and to learn the system. In

Component	*Nature*	*Status*
`:vxWorks`	Under vxWorks	
`:vxWorks:include`	*all* RT includes	
`:vxWorks:tape`	`tapeTask`, etc.	In progress
`:vxWorks:station`	`stnTask`, etc.	Prototype
`:vxWorks:archive`		Deferred
`:vxWorks:clock`		Deferred
`:vxWorks:job`	`jobTask`, etc.	Prototype
`:vxWorks:model`	`modlTask`	Done
`:vxWorks:calc`	GSFC library	Done
`:vxWorks:sched`		In progress
`:vxWorks:jobcontrol`		In progress
`:vxWorks:jobloader`	`jbldTask`	Done
`:vxWorks:hcb`	Device driver	Done
`:vxWorks:util`	Misc utilities	
`:vxWorks:util:efc`	Event flag library	Done
`:vxWorks:util:scripts`		
`:vxWorks:util:scripts:vsh`	Boot script	Prototype
`:vxWorks:util:scripts:tx`	CX script	`job1000.tx`
`:vxWorks:util:tables`	table library	Done
`:vxWorks:util:ctsk`	tasking library	Done
`:vxWorks:util:ctsk:test`		
`:vxWorks:tick`	`tickTask`	Done
`:sun`	Under SunOS	
`:sun:include`		
`:sun:lib`		
`:sun:bin`		
`:sun:etc`		
`:sun:hcb`	Compute tables	Done
`:sun:hcb:angles`		
`:sun:hcb:fringe`		
`:sun:dbms`	schema, batch jobs	Suspended
`:op-indep`	vxWorks & SunOS	
`:op-indep:include`		
`:init`	Initialize Delivery	Done
`:doc`	Documents	
`:doc:memo95`		History
`:doc:include`		
`:doc:misc_memos`		This memo

Table 1: NSE Component Structure

general, a "component" is a group of related source files plus the Makefile that contains the rules for processing them and delivering them to the operational code directory. For environment station1@dwells the delivery directories are /home/ccc/vlb/fxcorr/dwells/station1/{vxWorks,sun,doc}; the reader should be able to infer what directories new@Nse uses for delivery. An environment is an instance of a set of components; the instance can include the entire set (the whole system) or any subset of it. Programmers may own more than one environment.

We have chosen to make our directory structure correspond exactly to the NSE component structure, with a few exceptions (various include files). For example, the text of this document is source file jul90.tex for the "delivery" target of component :doc:misc_memos, and is in the directory /home/ccc/vlb/fxcorr/nse/doc/misc_memos.[1]

Individual files have versions (maintained by VCS [Version Control System], a typical checkout/putback utility); so do environments. There is a parent-child relationship between environments. Children are "acquired" from parents and "reconcile" new code back to the parent environment. Multiple children may reconcile changes to the parent, and pick up each other's changes by using the "resync" operation. A tool called the "resolver" aids in merging the changes to source files and file/component structures. In this sense our new environment exploits NSE technology to provide a communications mechanism for the orderly interchange of source code being developed in parallel by members of our Group. Recently a cycle of reconcile-resync operations led to two Group members sitting before a resolver window, which showed the multiple versions of their modules, and agreeing on how conflicting changes should be merged (non-conflicting changes were merged semi-automatically).

A parent environment may also be a child of another environment. In the case of our configuration, new is the child of release. So far we have not reconciled new back to release, because no version of the current experimental code has yet represented an operational version of the station "axis" of the project that the Group is currently working on. Probably releases will be accompanied by some sort of formal report of changes analogous to the monthly code change reports from the M&C Group.

[1]Our environments appear in the /home/ccc/vlb/fxcorr/nse directory for any process for which they are "activated", as though this directory were the mount point for a disk partition in ordinary Unix. Each process for which an environment is activated sees that environment's version of the file system mounted at that point (this is NSE magic, it depends on Sun's tfs [Translucent File System] device driver).

3.2 NSE Bugs

Our NSE administrator, J. Horstkotte, has become quite knowledgeable about the inner workings of NSE during the past two quarters. He has filed numerous bug reports with Sun. He has learned/developed work-arounds. We have been particularly frustrated by certain bugs involving file name changes, bugs which were exposed during the name changes involved in our decision to adopt a module naming convention. On two occasions Sun personnel have logged into CCC and fixed the database (NSE depends on a special kind of DBMS). Parallel development has not really been stopped, but members of the Group have had to work in their own NSE environments without reconciling and resyncing as frequently as we would like, and the integration of the remainder of our code and Group members into the NSE environments has been delayed more than we expected. Jim has not been able to work on application code as much as he or the rest of the Group desire. We anticipate receiving a new version (1.2.1) of NSE within a few weeks but it won't include fixes for our bug reports, and Sun have suggested that we enter into a "beta" relationship with them for their 1.3 release.

We are reluctant to recommend that other NRAO groups adopt NSE for group code management while these bugs exist in the product, although we ourselves believe that our project is still better off continuing to work around the bugs while awaiting the fixes because we have already paid the learning/workaround price. It appears to us that the product has real advantages, and that there is a good chance that the 1.3 release, perhaps to be available 1Q91, may be worthy of a solid recommendation.

4 New RT Hardware

Near the end of 2Q we received another Motorola MVME147 CPU. We have activated the second VME crate and now have two CPUs in each crate, two 147 CPUs in one and a 147 plus the older 131 CPU in the other. The Electronics Group periodically utilizes a CPU and MCB to check out DAR racks for stations, but otherwise the Correlator Group can use (and needs!) all four CPUs. The new CPU was purchased from our spares budget in order to support the Electronics Group requirement.

At present we have only one HCB controller, although plans and parts exist for a second. We have only one MCB. Consequently, test operation of RT code with actual hardware must be somewhat scheduled, and contends with engineering tests of the hardware. Functional testing of prototype boards has priority, of course.[2] We are still working out appropriate procedures for coordinating hardware and software testing, and software testing by multiple programmers.

[2]The FFT/MAC board test procedures are coded in C and utilize the same vxWorks HCB driver that will be used for the RT tasks. This driver and test code has been partially moved under NSE, in the `:vxWorks:hcb` component listed in Table 1.