NCCS - Scheduler
Design and Implementation

A. Götz

March 14, 1996

1 Introduction

The NCCS must provide support for the scheduling of multiple programs simultaneously while at the same time making sure that only one of them is executing at a given time. In the *NCCS - Design* document it was proposed that the above functionality be implemented in a program called the **Scheduler**. The **Scheduler** will keep track of which programs are scheduled, which program should run next and garantuee that it has exclusive access to all necessary resources.

This document deals with the **Scheduler**. It will describe the *design principles* on which the **Scheduler** will be based, the *data flow* and finally how it is planned to *implement* the **Scheduler**.

2 Design Principles

The Scheduler will be based on a number of principles. These principles will determine the functionality and interface the scheduler will offer to all users of the NCCS. The principles listed below are the result of discussions held between **A.Götz**, **M.Gaylard** and **J.Quick**.

Here is a list of the main design principles and what they signify :

1. **anti-trust** - the Scheduler will assume that the users of the NCCS **cannot** be trusted. That means that checks will be build into the system which prevent users from circumventing the Scheduler. Once the Scheduler has decided which is the next program to run it will inform all active parts of the NCCS. It is assumed that the active parts of the NCCS will be implemented as device servers. The device servers will check every incoming *write* request to prevent unauthorised programs from executing.

2. **scans** - all observations will be based on the concept of a **scan**. A scan will be the basic unit of observation. There will be a scan type per observation type. The type of observation will determine how the scan is executed e.g. track a single position, scan in declination (for mapping), do an on-off etc. Different scan types will be implemented depending on the different types of observation. Each observation type should be broken down into the smallest meaningful unit i.e. scan.

3. **scan id's** - the Scheduler will use the concept of **scan identification** to determine which program may execute at any time. The **scan id** will be a unique identifier which is used by the Scheduler to identify every scan. The scan type

will determine which resources are required by the observing program and inform the Scheduler. Resources which are not required by the scan type will be released by the Scheduler so they can be used by other programs.

4. **restricted resource super-class** - a new super-class, the **RestrictedResource class**, will implement the necessary methods for handling scan id's. Not all devices require to have restricted access. Those which do will be derived from the *RestictedResource* super-class. The super-class will keep track of the scan id, implement a command for receiving the new scan id from the Scheduler (using a special identifier to avoid other processes from posing as the Scheduler) and a method for checking the scan id.

5. **active scheduling** - the Scheduler will be **active**. It will periodically check the status of the current observing program, the list of suspended programs and background programs waiting to run. If there has been any change e.g. observing program has crashed, scan has ended/expired, a higher priority request has come in, the Scheduler will invalidate the present scan id and issue a new scan id to the appropriate program. Because the Scheduler is active it can abort a scan when a program has crashed or the scan has expired.

6. **observation cron table** - routine monitoring observing programs and background programs will be handled in a similar manner to to the way `cron` manages background tasks in Unix. Users will submit their background programs to a file. The file will contain all the necessary parameters for running the program at the right time(s). The Scheduler will periodically read this file and intepret its contents. The contents of the file will be very similar to the file SCHDL presently maintains on the HP.

7. **scheduling algorithm** - the Scheduler will have an algorithm for deciding which program to schedule next. It envisageable that multiple algorithms are implemented and that it is possible to switch between different algorithms at startup time or even on the fly.

8. **mono resource** - the Scheduler will allow only **one process** to have access to a restricted resource at a time, so-called *mono resourcing*. Other processes who want access to a resource have to wait for it to be freed. Mono resourcing does not mean mono tasking - multiple processes using different resources can run simultaneously.

3 Data Flow

The data flow for the Scheduler is depicted in *figure 1*. The diagram shows the various types of observing programs which will be run in various phases of execution.

The main observing program is *obs prog* which has requested a scan and had it granted. The Scheduler informs all required resources (device servers) e.g. steer and noise diodes, of the new valid *scan id*. This immediately invalidates the previous scan id.

Other programs will wait for end of the scan for the resource (e.g. the telescope) to be freed before they gain access to it. A process waiting for a resource (*queued obs prog*) can go into a sleep state waiting for the scheduler to wake it up or it can go off and do something and then request a new scan at a future date.

The Scheduler periodically wakes up (*periodic check* loop) and checks the validity of the current scan to see if it has terminated or expired. It also checks the list of background monitoring programs to see if any new programs have been added to

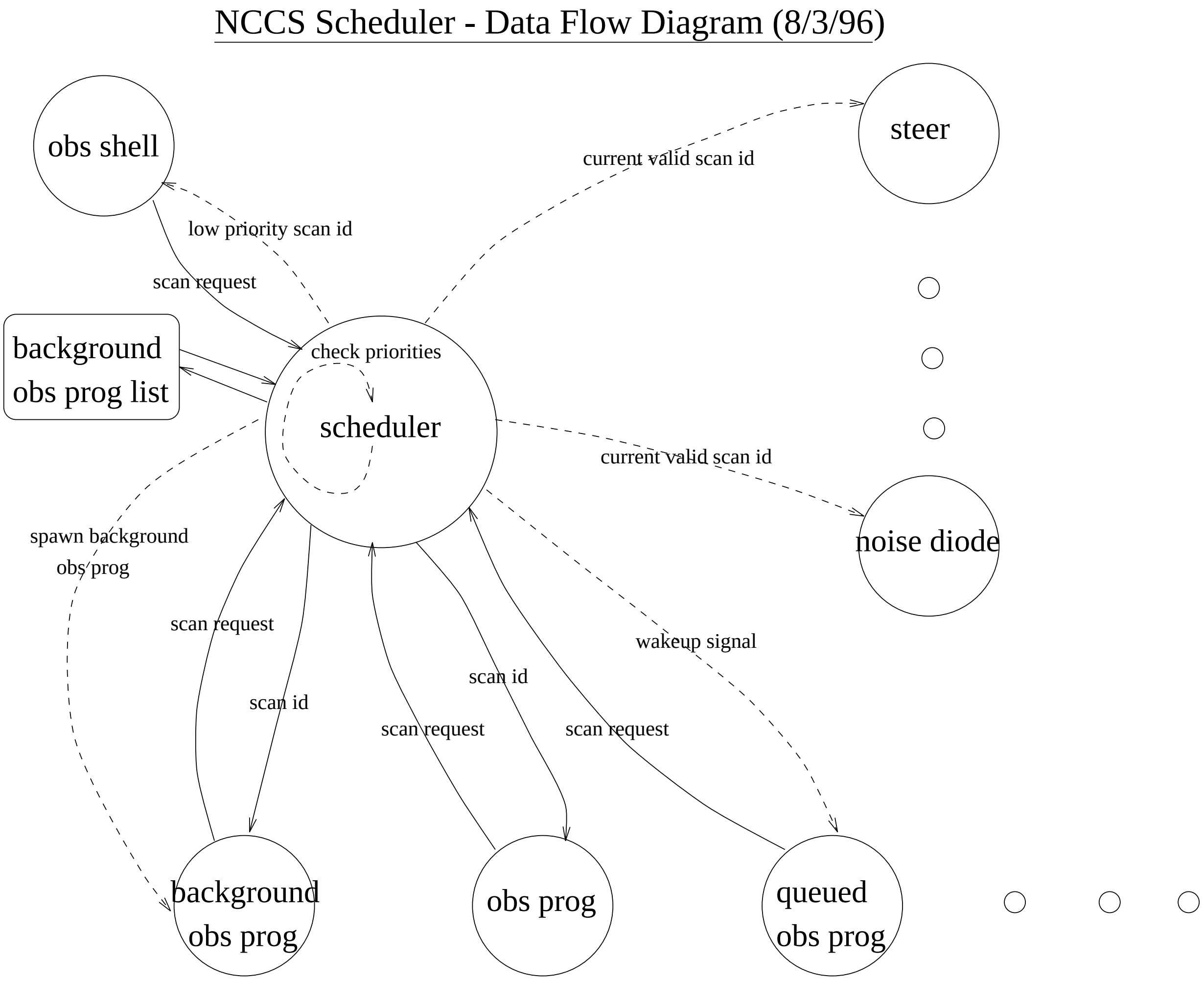

Figure 1: Data Flow Diagram of NCCS Scheduler

the list and if it is time to schedule one of the background programs. If one of the background programs is to be scheduled the Scheduler will spawn it and wait for it to start up. As soon as the program starts it should request a scan id and have it granted.

Finally there is an example of a interactive observing program (*obsh*) running at low priority which will have access to resources which are presently not being used.

4 Implementation

The Scheduler will be implemented as a **device server** with no hardware and only logic. Implementing the Scheduler as a device server means that it has a well-defined (and implemented) network protocol. The device server structure is well-defined and easy to understand for other device server programmers.

4.1 Commands

The Scheduler device server will implement the following commands :

1. `DevRequestScan` - to request a scan id.

   input parameters - type of request (blocking/non-blocking), type of scan, duration of scan, list of devices needed.

   output parameters - valid/null scan id .

2. `DevEndScan` - to end a scan.

   input parameters - scan id.

   output parameters - none.

3. `DevAbortScan` - to abort a scan.

   input parameters - scan id.

   output parameters - none.

4.2 Algorithm

The Scheduler has to decide how to allocate resources based on a predictable algorithm. The design of the Scheduler foresees the option to implement multiple algorithms and be able to switch between them. Two possible algorithms are described below.

4.2.1 Simple

The *simple* algorithm will be a straight-forward implementation. It will execute only the requests it receives. This includes the requests from routine monitoring programs. It will not try to optimise the use of the telescope. If there is a gap in the schedule it will not try to fill it.

4.2.2 Maximum Efficiency

The *maximum efficiency* algorithm will try to maximise the use of the telescope by being *pro-active*. If it sees the telescope is not being used it will actively schedule an observing program as a time filler.

4.3 File Format

Same as HP ascii file format.

5 Conclusion

Once this document has been approved implementation of the Scheduler can start.