Naval Research Laboratory

Stennis Space Center, MS 39529-5004

NRL/MR/7320--11-9353

Implementation of the Automated Numerical Model Performance Metrics System

JAMES D. DYKES

Ocean Dynamics and Prediction Branch
Oceanography Division

September 26, 2011

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
26-09-2011	Memorandum Report	

4. TITLE AND SUBTITLE

Implementation of the Automated Numerical Model Performance Metrics System

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
0602435N

6. AUTHOR(S)

James D. Dykes

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
73-4320-01-5

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Oceanography Division
Stennis Space Center, MS 39529-5004

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/7320--11-9353

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
One Liberty Center
875 North Randolph Street, Suite 1425
Arlington, VA 22203

10. SPONSOR / MONITOR'S ACRONYM(S)

ONR

11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Known as AutoMetrics, this set of software modules was developed to provide model-observation and model-model comparison matchups and statistics to help modellers and forecasters assess the performance of the ocean circulation models, primarily the Navy Coastal Ocean Model and the Hybrid Coordinate Ocean Model. This system is fully automatic and gives users much flexibility in what model runs to be evaluated by this system, in real-time operations or for long-term historical model runs.

15. SUBJECT TERMS

Model performance
Model performance statistics
Automated model performance metrics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			James D. Dykes
Unclassified	Unclassified	Unclassified	Unclassified Unlimited	31	19b. TELEPHONE NUMBER *(include area code)* (288) 688-5787

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

This page intentionally left blank.

Table of Contents

Abstract

Known as AutoMetrics, this set of software modules was developed to provide model-observation and model-model comparison matchups and statistics to help modelers and forecasters assess the performance of the ocean circulation models, primarily the Navy Coastal Ocean Model and the Hybrid Coordinate Ocean Model. This system is fully automatic and gives users much flexibility in what model runs to be evaluated by this system, in real-time operations or for long-term historical model runs.

1. Introduction

The AutoMetrics system was the result of a project under the Rapid Transition Program to transition to NAVOCEANO software that provided information to the oceanographers so that they can assess the performance of numerical ocean prediction models run at NAVOCEANO. The Navy Coastal Ocean Model (NCOM) (Rowley et al., 2002; Martin, 2000) was initially the primary target, but also the Hybrid Coordinate Ocean Model (HYCOM) (Wallcraft, 2007) can be used in this system as well. In fact, any ocean circulation model output in the format described below can be used in AutoMetrics.

Primarily ocean profiles of observed temperature and salinity are collected and matched with profiles of modeled same parameters corresponding to same location and time. This also applies to surface –only observations. The relationship between observations and model output consists of one-to-many, because from a given model domain and its several runs there exists multiple forecasts for a given time and place. To further complicate matters, the availability time of both observations and model output are considered arbitrary and the system must continually check for both to provide the most up-to-date matches in a timely manner. Ultimately a data base is built with a set of matches that can be analysed statistically in a variety of ways.

Based on the observed parameters, sound speed is calculated using the observation profiles and the associated modelled profiles, resulting in matches of sound speed profiles. From that RP33 (NAVOCEANO, 1999) parameters including sonic layer depth (SLD) and below-layer gradient (BLG) are computed forming additional sets of matches, all of which can be statistically analysed. More RP33 parameters are being considered for addition.

For the assessment of model performance that do not involve comparisons to observations there are procedures which take differences between any two fields of like parameters. A simpler implementation is used for fields of like domain size which is useful to compare a current analysis or prediction to a previously predicted field for the same target time. A more involved routine can handle two fields on different grids.

Also, described here are other utilities to be used to check the data. Scripts were formulated for refitting into a general automation system which is also described.

For readers of this document a working knowledge of UNIX-like operating systems is assumed.

Manuscript approved August 31, 2011.

2. Input and output data files

2.1 Model Prediction Output

These files typically from NCOM or HYCOM are netCDF (network Common Data Format)[1] files whose arrangement follow a convention prescribed by NAVOCEANO and augment the NetCDF COARDS convention[2] which is registered with Unidata and is available at the University Corporation for Atmospheric Research (UCAR).

Files consist of predictions up to 96 hours or more. All the prediction hours or TAUs may be contained in one file or in separate files. The file naming convention as specified by NAVOCEANO consists of terminology for the type of model, the domain over which the model ran, a 10-digit date-time group and, if applicable, the one TAU. The date-time group consists of a four-digit year, a two-digit month, a two-digit day of the month, and a two-digit hour of the 24-hour clock. All the terms in the name are connected with underscores. An extension, *.nc*, is attached. An example of such a file follows:

```
ncom_relo_okrwtrgh_v4_sub1_2009051100_t012.nc
```

This file is output from a regional NCOM as indicated by *ncom_relo* run over a domain called *okrwtgh_v4_sub1*. No particular number of terms that are delimited by the underscores is required. The cycle time (or run time) was 11 May 2009, at 00 GMT. The one prediction contained in this file is at TAU 12 or the 12th forecast hour from the cycle time. Other examples of combinations of model and domain in the terminology in the prefix of the file name include *ncom_glb_regp01* and *hycom_glb_908_reg01*, which are cut-outs from model runs that covered the global domain.

2.2 Observations

The data in these files come from satellites, ships, XBTs, gliders, and ARGO floats and are first processed through OcnQC, a component of the Navy Coupled Ocean Data Assimilation (NCODA) system (Cummings, 2005) to form the files used in the matchup utilities. A version of the NCODA OcnQC system operates on-site at NAVOCEANO (Lunde and Coelho, 2010) providing files locally in real-time processed from a data feed. Observational data accumulate in each 24-hour period file, during and even beyond the24-hour period as data is received.

The files with the *.profile*, *.ship*, and *.altim* extensions are used in this software package. The root part of the name is a 10-digit date-time group with the same layout as described above. Observations contained in the files were taken within the 24-hour period after the date indicated by the 10-digit date-time group. The *.profile* files contain, as the name implies, observation profiles of temperature and salinity among other things. The *.ship* files are focused on the sea surface temperatures. The *.altim* files are all about the sea surface height anomalies. Data times in all these files always start at 00 GMT.

[1] University Corporation for Atmospheric Research—Unidata NetCDF. http://www.unidata.ucar.edu/netCDF.
[2] University Corporation for Atmospheric Research—NetCDF Conventions. http://www.unidata.ucar.edu/netCDF/Conventions.html.

The data in the files are in binary structure written by FORTRAN programmes. Software functions written for this system are described later in this report showing how these files are read.

2.3 Matchup files

These files are results from the *MatchUp* utility described below. They consist of the pairs of observations and model output (or derived components) in a binary structure which includes metadata.

3. Software components

The components described here consist of stand-alone utilities and the functions that are included in these utilities. This is not a full software design description.

For all the components listed below the same pattern for compiling the code is followed assuming that a UNIX-type operating system is used. Source code for the utilities are organised in each of their related named subdirectories in *$ARTP_HOME/src*, where *$ARTP_HOME* is the root directory for all the software discussed in this document. The source code for the functions reside in *$ARTP_HOME/libsrc*. Each directory has a *Makefile* in which is included a *conf-[Cray-XT|IBM|Linux]* file which contains architecture specific settings. Where applicable, the path to the netCDF library file must be included in the *Makefile*. Compiled utilities get installed in *$ARTP_HOME/bin* and the library is *$ARTP_HOME/lib/libartp.a* which must be built first. The interfaces for the functions are in files located in *$ARTP_HOME/include*.

After editing the *Makefile* for the appropriate *conf-* file*, checking paths, and compiling the AutoMetrics library, the command *make* executed in the directory where the utilities reside will compile the utilities in that directory right away. Executing *make install* will put the executables into *$ARTP_HOME/bin.*

Source code is updated on the NRL SVN server. Below are the utilities described in sections named just as they are organised in directories for the source code.

3.1 MatchUp

The one utility is called *matchup_drvr.x* and matches OcnQC data to parameters of multi-dimensional model output, up to four dimensions. Two observation file types can be used: *.ship* and *.profile* (*.altim* not at this time). Usage is as follows:

```
% matchup_drvr.x list_ncfiles list_obfiles mafile
```

The first argument, *list_ncfiles* is a file containing a list of netCDF files. The second argument, *list_obfiles*, is a list of observation files to be operated with the netCDF files in the first list. For either of these lists of files, absolute or relative directory paths can be used. The final argument, *mafile*, is the binary, direct access output file with the resulting matchups. Although there is no technical limit to the number of files that can be listed in either list file, in practice there are limits in time and memory.

The programme goes through the list of netCDF files and for each one tests for matches to the observations stored in the files listed in the other list file. Naturally, many times the programme loops through data with nothing to match where a netCDF file does not cover the time frame of the observation file. Since the relationship between observations and forecasts is one-to-many, there will be many redundant instances of any observation as they are matched and saved together with each forecast for the same target time and place.

There are two ways to treat finding the time of the data in the netCDF file to match the time of the observation. If a netCDF file contains one time dimension, i.e. a single TAU, then automatically *matchup_drvr.x* compares observations at their measurement times to a 1.5 hour window around the TAUs in the netCDF file to form the matchup if within the window. Otherwise, the times are interpolated between TAUs in the same netCDF file. If the latter method is preferred and the netCDF files to be used all contain single TAUs, then they can be combined with the *ncrcat* command which is part of the NCO (netCDF Operators) (Zender, 2007) software package. Often due to file sizes, files may only be combined to as many as two TAUs at one time.

Interpolation is linear along horizontal space within the fields of model output and as stated above may also be so along time if more than one TAU is present. Vertically, interpolation is done on a piece-wise Hermite curve along the profile of an observation resulting in data pairs at each modelled level. The matchup pairs are contained in a record with the following structure:

```
float btm;          // bottom depth
float lat;          // latitude of observation
float lon;          // longitude of observation
int nlvls;          // number of levels
char parm[12];      // name of the parameter
float *lvl;         // array of levels (depths for ocean) based on model
float *obm_val;     // observed value at each level
float *mdm_val;     // model value at each level
float *diffm_val;   // difference – modelled minus observed
char ob_dtg [12];   // date-time group of the time of observation
char md_dtg [12];   // date-time group of the time of model value
double tau;         // forecast hour since model cycle start
char sign[72];      // call sign/name of the observation platform
char id[72];        // unique identifier of observation
char model[72];     // name of the model with region
```

For each cycle in a model run, many records may be stored in one file. Many cycles of a model may be combined together which may result in very large files. A header four bytes long is placed at the beginning of each file. This is a float value indicating the version of the data structure used, which helps interpreting software distinguish between older and new, improved versions of this structure.

3.2 OcnObs

A utility called *ocnobs.x* reads the OcnQC files *(.ship*, *.profile*, or *.altim*) and returns contents to *stdout*. This is a handy way to dump out the contents of a file for verification. Usage follows:

```
% ocnobs.x obsfile | more
```

The argument is an observation file. It is advisable to pipe the *stdout* to *more* or redirect it to a file.

3.3 Stats

The main one of two utilities in this section is called *stats.x* and computes statistics from a set of matchup files resulting from the *matchup_drvr.x* reporting results in a table in *stdout*. Usage follows:

```
% stats.x list_mafiles parameter TAU1 TAU2
```

The first argument, *list_mafiles*, is a list of matchup files. The lists of files may include absolute or relative directory paths. The second argument is the parameter of interest, which may include values like, *water_temp*, *salinity*, *sound_speed*, *SonicLayerDepth*, and *BelowLayerGradient* as precisely spelled out. The last two arguments specify the beginning and ending TAUs in which the desired statistics are to be computed. An example of a table output follows:

```
stats.x: opening matchup file: ../../data/matchups/daily_0_72_profile_ncom_relo_wpac_2_u-2011051600-interp.matchup
stats.x: reading matchup file
Version: 1
stats.x: all matches read.

 nc_level       MB     RMSD   StdDev        R  mdlmean  obsmean    Bcond  Buncond       SS        N
      0.0    0.155    1.483    1.476    0.979   23.869   23.714    0.002   66.281  -65.324      480
      2.0    0.112    1.480    1.478    0.979   23.721   23.609    0.001   35.491  -34.532      467
      4.0    0.150    1.574    1.569    0.977   22.847   22.697    0.000   68.385  -67.430      407
      6.0    0.159    1.561    1.555    0.978   22.823   22.664    0.000   77.224  -76.269      407
      8.0    0.171    1.546    1.538    0.978   22.791   22.620    0.001   89.650  -88.694      407
     10.0    0.174    1.536    1.528    0.979   22.789   22.615    0.001   92.920  -91.963      405
     12.0    0.186    1.530    1.520    0.979   22.742   22.557    0.001  107.530 -106.572      405
     15.0    0.197    1.558    1.548    0.979   22.662   22.465    0.001  122.655 -121.699      404
     20.0    0.189    1.578    1.568    0.978   22.488   22.299    0.002  116.202 -115.247      402
     25.0    0.160    1.589    1.583    0.978   22.237   22.078    0.002   85.117  -84.162      401
     30.0    0.151    1.593    1.587    0.978   21.994   21.843    0.002   76.689  -75.734      399
     35.0    0.142    1.601    1.596    0.978   21.779   21.637    0.002   68.185  -67.231      395
     40.0    0.142    1.610    1.606    0.977   21.518   21.376    0.002   66.826  -65.873      393
     45.0    0.127    1.604    1.601    0.977   21.285   21.159    0.001   50.816  -49.862      389
     50.0    0.104    1.648    1.647    0.975   21.034   20.931    0.000   33.612  -32.661      385
     60.0    0.067    1.720    1.721    0.972   20.461   20.394    0.000   13.039  -12.095      380
     70.0    0.051    1.739    1.740    0.970   19.935   19.884    0.001    6.594   -5.655      373
     80.0    0.094    1.757    1.757    0.968   19.325   19.231    0.004   20.015  -19.081      363
     90.0    0.105    1.764    1.764    0.966   18.709   18.604    0.006   21.524  -20.596      357
    100.0    0.185    1.756    1.748    0.965   18.191   18.007    0.008   60.482  -59.559      344
    125.0    0.192    1.712    1.704    0.964   16.728   16.536    0.003   58.870  -57.943      316
    150.0    0.152    1.548    1.543    0.968   15.120   14.968    0.000   32.546  -31.609      296
    200.0    0.106    1.287    1.285    0.975   12.862   12.756    0.000   11.962  -11.013      278
    250.0   -0.061    1.155    1.156    0.978   11.125   11.186    0.000    3.363   -2.407      266
    300.0   -0.072    0.911    0.910    0.985    9.754    9.826    0.001    3.944   -2.975      257
    350.0   -0.056    0.772    0.771    0.987    8.953    9.009    0.000    1.686   -0.711      255
    400.0   -0.009    0.710    0.712    0.987    8.323    8.332    0.000    0.030    0.944      252
    500.0    0.044    0.563    0.562    0.988    7.168    7.123    0.007    0.275    0.694      246
    600.0    0.098    0.459    0.449    0.988    6.184    6.086    0.019    0.438    0.518      239
    700.0    0.105    0.446    0.435    0.978    5.500    5.395    0.051    0.115    0.790      232
    800.0    0.232    0.488    0.432    0.932    5.445    5.213    0.124    0.062    0.683      124
    900.0    0.119    0.319    0.298    0.947    4.624    4.505    0.023    0.008    0.865      103
   1000.0    0.030    0.204    0.203    0.944    3.843    3.813    0.003    0.000    0.888       49
   1250.0    0.012    0.127    0.128    0.935    3.067    3.055    0.050    0.000    0.825       39
   1500.0    0.028    0.093    0.090    0.906    2.577    2.548    0.033    0.000    0.788       33
   2000.0   -0.015    0.039    0.039    0.875    2.024    2.038    0.372    0.000    0.394        7
   2500.0   -0.007 -999.000 -999.000 -999.000    1.748    1.755 -999.000 -999.000 -999.000        1
   3000.0    0.047 -999.000 -999.000 -999.000    1.679    1.632 -999.000 -999.000 -999.000        1
stats.x: summary file written.
```

Labels of each column and their meaning follow:

nc_level – netCDF model level in meters, depth for ocean circulation models
MB – mean bias or mean difference, model minus observation
RMSD – root mean squared difference
StdDev – Standard deviation of differences.
R – correlation coefficient
Mdlmean – mean of modelled values
Obsmean – mean of observed values
Bcond – conditional bias
Buncond – unconditional bias
SS – skill score
N – number of points

The formulations for *RMSD*, *R*, *Bcond*, *Buncond*, and *SS* are computed and used as described in Bara et al., 2006.

A subset of *stats.x* is *rd_matchup.x* which simply reads a match-up file and dumps the contents to *stdout*, which is useful for verifying the performance of *matchup_drvr.x.* It also provides the exact spelling of the parameters which can be used as an argument for stats.x. The usage is much simpler:

```
% rd_matchup.x mafile | more
```

where *mafile* is the matchup file that was generated by *matchup_drvr.x.* It is advisable to pipe the *stdout* to *more* or redirect it to a file.

3.4 CompareModels

A utility called *compare_models.x* takes the differences between two different models of fields of like parameters. Usage follows:

```
% compare_models.x ncfile1 ncfile2 [dtime]
```

The first two arguments are the two netCDF model output files with which a difference is taken, *ncfile1* minus *ncfile2*. These two files do not need to have the same resolution domain grids, but they should be covering the same area, but not necessarily exactly. The optional *dtime* argument is the increment in time desired. The default is that all times are done. The resulting netCDF files are named with a combination of the names of the input files plus an additional extension tacked on.

4. Automated System

4.1 Run Environment

The running system described here is set up using these environment variables:

```
$ARTP_HOME    ~someone/models/ARTP
$ARTP_OPS     $OPS/models/ARTP
$LISTS        $ARTP_OPS/lists
```

```
$LOGS          $ARTP_OPS/logs
$LOCKS         $ARTP_OPS/locks
$PATH          ${PATH}:${ARTP_OPS}/bin
```

See Appendix A, General Automation, for information on how the general environment is applied and where variables such as *$OPS* are defined. The *$ARTP_HOME* is in *~someone*, a home directory, likely to be */u/home/ooc* for NAVOCEANO, where the AutoMetrics system permanently resides. The *$LISTS*, *$LOGS*, and *$LOCKS* directories are set for convenience, but they otherwise can remain their defaults in accordance with the general automation system setup. After setting up your own general environment, the user can set up an AutoMetrics run environment like this:

```
% source ~someone/models/ARTP/etc/setup.csh
```

when in tcsh, or csh shell. And for for sh, ksh, or bash shell:

```
% . ~someone/models/ARTP/etc/setup.sh
```

Along with the environment set up, user directories and links are made in *$ARTP_OPS*. This environment will be expected and referenced by the AutoMetrics system. System directories that are used:

```
$ARTP_HOME/bin        AutoMetrics-specific scripts and executables
$ARTP_HOME/etc        set-up info files and templates
$ARTP_HOME/lists      do- and done-lists for daily automation
$ARTP_OPS/logs        log files for automation
$ARTP_OPS/locks       lock files for automation
$ARTP_OPS/data        model and observational input data and results
```

For each of these directories on *$ARTP_HOME* and *$ARTP_OPS* is a corresponding link from the other directory result, in effect, the entire structure available on both sides. Consider the content on *$ARTP_HOME* to be permanent data and that on *$ARTP_OPS* temporary/scratch data. For the list above, the first three physically reside in the permanent space whilst the last three reside on the scratch directory where files are scrubbed after a few days. Note that on the archive machine where the results are stored the same structure exists as in the scratch directory of the computational machine except that the storage is permanent, of course.

4.2 AutoMetrics Scripts

Two main scripts are implemented in the automated system at NAVOCEANO: 1) *matchup_set* which submits a PBS batch job script, *matchup_agent.job* which runs *matchup_drvr.x* ; and 2) *ncdiffr* which takes the difference between two model output fields of same-size mesh using NCO utilities. All these scripts are listed in Appendix B.

These script names are really links to scripts with the *.sh* extenstion located in *$ARTP_HOME/bin*. The arguments are *$dtg*, a 10-digit date-time group of the cycle run, and *$item* consisting of the domain and other pieces of information. This argument list complies with the requirements of the general automation system.

Here is an example of running on the command line *matchup_set* for cycle time 2010 April 01 00 GMT for the regional NCOM domain, *ncom_relo_wpac_2_u*:

```
% matchup_set 2010040100 ncom_relo_wpac_2_u:0:3:72
```

working from TAU 00 through 72 every 3 hours. The second argument is a combination of parameters that will be parsed in the script. When this script is successful, a job is submitted, after which a file, *jobdone-$dtg*, indicating completion is written. When this script attempts to run for this particular task and date again, it will see this file and call it a successful completion for the purposes of the general automated system. Note that it is possible for models to run at times other than 00 GMT e.g. 06, 12, or 18 GMT, in which cases the script knows to grab the first 00 GMT observation file and others after to cover the time frame in question.

As of this writing, the DSRC IBM AIX machines DaVinci and Pascal, and the Cray XT Einstein all use the PBS batch queuing system for submitting jobs.

The *$item* input for *ncdiffr* consists of a colon-delimited set of additional numbers appended to the domain which is parsed and used to control the TAUs used in making the difference, e.g.:

```
% ncdiffr 2010040100 ncom_relo_anex32:24:48
```

where the difference will be made between fields for TAU 24 of cycle 2010040100 and TAU 48 2010033100, i.e. two fields for the same valid time of 2010040200. This is an example of a typical comparison desired by the modelers at NAVOCEANO. Note that this script, though it has a similar role, is not the same software as, nor does it use, the utility, *compare_model.x*, described above.

4.3 Real-time and Hindcast Automation

The *matchup_set* and *ncdiffr* scripts are run by a *$ARTP_HOME/bin/cronpanel*. Located in *$ARTP_OPS/logs* are *loop_mgr* log files for the corresponding instances of these scripts. The following is partial listing of a sample *cronpanel*:

```
# Matchups and differences for NCOM
export days_back=1
export instance=matchup
run_lock loop_mgr matchup_set &
export instance=ncdiffr
run_lock loop_mgr ncdiffr &
```

Appendix A on General Automation describes how this application is set into motion for each *$item* and *$dtg* for automation. To apply the *loop_mgr* utility the user can add an entry into the do-list, e.g. *$LISTS/do_matchup*, as in this example:

```
ncom_relo_wpac_2_u:0:3:72 00 24
ncom_relo_socal_u:0:3:72 00 24
ncom_relo_useast_u:0:3:72 00 24
ncom_relo_amseas:0:3:72 00 24
ncom_relo_ecskur:0:3:72 00 24
ncom_relo_fukushima_1km_tmp:0:3:72 00 24
ncom_relo_sendai:0:3:72 00 24
ncom_relo_spss:0:3:72 00 24
#ncom_relo_psea:0:3:72 00 24
```

One item is commented out with #, so it is ignored. The two numerals following each *$item* control the start time and period of time between each run. Without those the default is 00 and 12. Normally, models run starting with 00 GMT and every 12 hours, but NCOM is an example of 24-hourly runs.

In addition to the do-lists in the *$LISTS* directory are the done-lists that keep a record of all items for completed cycles and the time of completion. For the *matchup_set* and *ncdiffr* scripts are the lists, *do_matchup* and *done_matchup*, and *do_ncdiffr* and *done_ncdffir*, respectively.

Model output in real-time from OOC that will be used in the scripts described above are located in various directories as prescribed by the model operations and their working file systems. Locally, netCDF files can be thrown into the hopper at *$ARTP_OPS/data/netCDF*. Observational profiles in real-time from NAVOCEANO operations are located in */u/home/ooc/models/ncoda/etc/$dist/$filetype* where *$dist* is either *navoqc_public*, *navoqc_rstrct*, or *navoqc_secret, and $filetype* is either *profile, ship,* or *altim*. These same types of data are also available on the GODAE server, http://www.usgodae.org/ftp/outgoing/fnmoc/data/ocn/. Locally, observational files can be thrown into the hopper at *$ARTP_OPS/data/ocnqc*.

Directories are created in *$ARTP_OPS/run* for each *$item* on the list named the same. Their log files and various work files pertaining to the *$item* are created and used within these work directories.

Results from *matchup_agent.job* arrive at *$ARTP_OPS/data/matchups*. Results from *ncdiffr* arrive at *$ARTP_OPS/data/model-model*. Monthly tar files are accumulated in *$ARTP_OPS/data/tarfiles.* These files end up on the archive machine which as of this writing is Newton.

Here is an important note regarding real-time matchups and completeness of data sets. Since observational files are accumulating during the 24-hour period of that file and beyond even for days, the most complete set of matchups is collected only after a significant period of time after initial model cycle time. If more current sets of matchups were needed, then users would have to accept the possibility of missing many observations that had not been received yet. If continual updates are called for, then deleting the done-list after every run will allow the scripts to process the same matchups over and over allowing for new observations to get included as they are received, though this is considered "brute force". Naturally, this is not an issue for "hindcast" runs.

Running in "hindcast" mode means rerunning for a period of time in the past. The cronpanel and/or the do-list would need to be set up with the date in the past and the time beyond. For example, if matchups are required for the month of July 2011, then entries in the cronpanel might look like this:

```
# Rerun for matchups and differences for NCOM
export dtg=2011070100
export days_for=31
export instance=matchup_rerun
run_lock loop_mgr matchup_set &
export instance=ncdiffr_rerun
run_lock loop_mgr ncdiffr &
```

In this example the do-lists and done-lists would be named *do_matchup_rerun*, *done_matchup_rerun*, *do_ncdiffr_rerun*, and *done_ncdiffr_rerun*. The value of *instance* variable and the resulting name of the do- and done-lists are up to the user.

Or, leaving the cronpanel as originally set up for real-time, the do-list (like the example listed above) can be edited with additional entries on each line indicating the date start and the days of duration as follows:

```
ncom_relo_wpac_2_u:0:3:72 00 24 2011070100 0 31
ncom_relo_socal_u:0:3:72 00 24 2011070100 0 31
ncom_relo_useast_u:0:3:72 00 24 2011070100 0 31
ncom_relo_amseas:0:3:72 00 24 2011070100 0 31
ncom_relo_ecskur:0:3:72 00 24 2011070100 0 31
```

```
ncom_relo_fukushima_1km_tmp:0:3:72 00 24 2011070100 0 31
ncom_relo_sendai:0:3:72 00 24 2011070100 0 31
ncom_relo_spss:0:3:72 00 24 2011070100 0 31
#ncom_relo_psea:0:3:72 00 24 2011070100 0 31
```

Everything else functions the same. The done-list will keep the system from redoing those items for those times as before.

4.4 User Hints

Usually the system can run automatically without user intervention. But, there are a few things that may help users to diagnose the situation if something goes wrong, and usually it is for some external reason. If there are no results showing up, check that the computer system is functioning as it should such as the cron and the batch queuing system. Check that the file system is in place, possibility deleted in the /scr directory or not mounted. Make sure the paths to executables and libraries are in place. This is important when relying on a general environment where commands like *timecalc* and *add_digits* on which so much relies might become unavailable.

A good place to start in the AutoMetrics system itself is looking at the log files. The files in *$ARTP_OPS/logs* are indicative of the behaviour of the *matchup_set* or *ncdiffr* scripts and may point out some deficiencies which again are external to AutoMetrics, such as data not arriving as it should. The *log-ARTP* file in *$ARTP_OPS/run* for each *$item* indicate the behaviour of the queued job and the various long-named log files show what the *matchup_drvr.x* programme is doing.

Also, within those *$item* directories are the *jobrunning* and *jobready* files indicating that there is a job submitted in the queue that is either running or waiting ready to run, respectively. If there is a *jobrunning* file whose *jobid* contained in that file does not exist in the queue, the automated system knows to replace the *jobrunning* file with the correct one the next time around. But, this not the case for the *jobready* file, so the removal of this file is necessary for it to resume for that *$item*. When the task is done for a particular *$item* and *$dtg*, then a *jobdone-$dtg* file is written which *matchup_set* recognizes and calls this task done for the automated system to list in the done-list.

Progress can be easily monitored by checking out the *$ARTP_OPS/tarfiles* directory. There each tar file contains the running total of all the matchups made for the month in a model.

Acknowledgements

This project was funded by ONR Program Element 0602435N. Thanks go to NAVOCEANO personnel for their requirements inputs and valuable feedback. Philip Fanguy from QinetiQ North America and Josie Fabre from the Naval Research Laboratory contributed to the source code of the utilities. Thanks go to Rachel Bourg at NAVOCEANO for her assistance setting up the software system on all the operational machines and assuring their proper functioning.

References

Kara, A.B., C.N. Barron, P.J. Martin, L.F. Smedstad, and R.C. Rhodes, 2006: Validation of interannual simulations from the $1/8^0$ global Navy Coastal Ocean Model (NCOM), *Ocean Modelling 11*, 376-398.

Cummings, J.A., 2005: Operational multivariate ocean data assimilation. *Q J R Meteorol. Soc 131*, 3583–3604.

Hodur, R.M., 1997: The Naval Research Laboratory's Coupled Ocean/Atmopheric Mesoscale Prediction System (COAMPS). *Mon. Wea. Rev*., *125*, 1414-1430.

Lunde, B. and E. F. Coelho, 2009: Implementations of the Navy Coupled Ocean Data Assimilation System at the Naval Oceanographic Office. *Marine Tech. Soc.*, Oct. 26-29, Oceans 2009, Biloxi, Mississippi.

Martin, P.J., 2000: Description of the Navy Coastal Ocean Model Version 1.0. NRL Formal Report, NLR/FM/7320--00-9962. Naval Research Laboratory, Stennis Space Center, Mississippi, 45 pp.

Naval Oceanographic Office, Code N72, Claimancy Training Division, Tactical Support Branch, 1999: Fleet Oceanographic and Acoustic Reference Manual, RP33, April 1999, 216 pp.

Rowley, C., C. Barron, L. Smedstad, R. Rhodes, 2002: Real-time ocean data assimilation and prediction with global NCOM, *Marine Tech. Soc.*, Oct. 29-31, Oceans 2002, Biloxi, Mississippi.

Wallcraft, A., H. Hurlburt, E.J. Metzger, E. Chassignet, J. Cummings, and O.M. Smedstad, 2007: Global Ocean Prediction Using HYCOM, *HPCMP-UGC*. 2007 DoD High Performance Computing Modernization Program Users Group Conference, 259-262.

Zender, C.S., *NCO User's Guide*, Version 3.9.3 (2007) Available from: http://nco.sf.net/nco.pdf (1, 3, 3.3).

Appendix A – General Automation System

This system provides general purpose tools and a general way to automatically run applications on a cycle. Applications are typically scripts that prepare for, run, and process output for numerical models that run every day.

A.1 General Environment

The system described here is set up using these general environment variables:

```
$USR            ~someone/usr
$OPS            $scratch/someone
$ETC            $USR/etc
$LIB            $USR/lib
$BIN            $USR/bin
$LISTS          $USR/lists
$LOCKS          $OPS/locks
$LOGS           $OPS/logs
$WORK           $OPS/work
$PATH           $PATH:$BIN
```

$OPS is created for the user (*~someone*) in some temporary space on scratch directories which could be */tmp* or */scr* whilst *$USR* is located permanently at *~someone*, a home directory. The user can set up the environment like this:

```
% source ~someone/usr/etc/setup.csh
```

when in csh, or tcsh shell, And for in sh, ksh, or bash shell use:

```
% . ~someone/usr/etc/setup.sh
```

Along with the environment set up, user directories and links are made in *$OPS*. This environment will be expected and referenced by the automation system. There is facility to allow a different user use *someone's* environment as their own and implement the software in *someone's* user account.

A.2 Looping Manager

Figure 1 illustrates how *loop_mgr.pl* (linked to *loop_mgr*), a Perl program, is used to automate applications on a cyclical basis. For example, COAMPS®, a numerical meteorological model operationally run at FNMOC (Hodur, 1997), can be run for certain domains every 12 hours. Any application used by *loop_mgr* would be required to take two arguments: 1) a 10-digit date-time group consisting of four digits for the year, two digits for the month of the year, two digits for the day of the month and two digits for the hour of the day (YYYYMMDDHH); and 2) a unique item identifier which can be parsed in a script. For example, a script that sets up and runs COAMPS®, *coampsset*, could take the arguments: 2007040100 Adriatic:48, which *loop_mgr* would formulate from checking the do-list and the done-list. The second argument in this example might have some delimiter, a colon in this case, which could be parsed by a script designed to parse for a domain name, Adriatic, and a forecast period, 48.

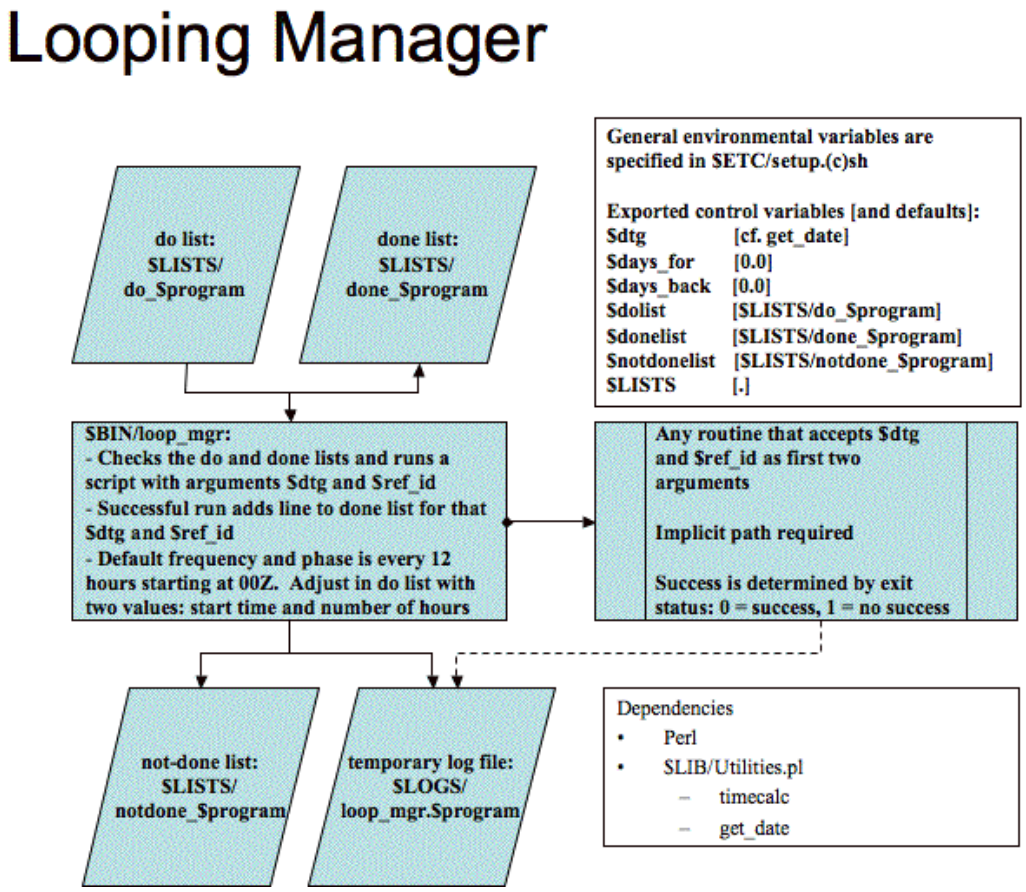

Figure 1 Diagram of Looping Manager Procedure.

It is cleaner to keep applications named without an extension, since the names of the do- and done-lists are named after the application by default. Note that the general environment variables may be used as-is or some can be redefined to confine the work someplace else. Here is an example command sequence (in a "sh" script kicked off in a cron) to run COAMPS® in real-time without checking back or forward in time:

```
export days_for=0.0
export days_back=0.0
loop_mgr coampsset
```

To override the current real-time date for a 6-month hindcast starting at 01 Jan 2010 00 GMT:

```
export dtg=2010010100
export days_forward=181.0
export days_back=0.0
```

The do-list, *do_coampsset*, could contain these sample entries:

```
Adriatic:48 00 12
Gulf_of_Mexico:72 00 12
```

where the second and third entries are offset time and frequency both in hours, these values being the defaults.

When the application being applied by *loop_mgr* exits with return 0, then this job is considered done and added into the done-list. No more attempts will be made to apply the application for this *$dtg* group and *$item* unless that line item is removed and the *$days_back*, *$days_for*, and *$dtg* are set to allow that time to be attempted again. The *$do_lists* and *$done_lists* variables can also be defined over the defaults.

Another way to do the hindcast runs is to provide these additional entries in the do-list as follows:

```
Adriatic:12 00 12 2010010100 0 181
Gulf_of_Mexico:12 00 12 1997010100 0 181
```

which has the same effect as assigning the environment variables *$dtg*, *$days_back*, and *$day_for*, respectively, the same added entries in this do-list example.

Listing of the *loop_mgr* Perl script is as follows:

```
#!/usr/bin/perl -w
# See documentation below

use Env;
require "$USR/lib/General.pm";

$program = $ARGV[0];

if ($ENV{"dtg"}) {
        $curr_dtg = $ENV{"dtg"};
        print "loop_mgr: Using dtg = $curr_dtg\n";
} else {
        $curr_date = get_date ();
        $curr_dtg = "$curr_date"."00";
}
system( "date" );

# Define control variables from the environment or use defaults
$instance = definevar( "instance", "$program" );
$LISTS = definevar( "LISTS", "." );
$dolist = definevar( "dolist", "$LISTS/do_$instance" );
$donelist = definevar( "donelist", "$LISTS/done_$instance" );
$notdonelist = definevar( "notdonelist", "$LISTS/notdone_$instance" );
system ("touch $notdonelist");
system ("rm $notdonelist");
system ("touch $notdonelist");

$days_back = definevar( "days_back", 1 );
$days_for = definevar( "days_for", 0 );
$lsequence = definevar( "lsequence", "true" );

# Loop for each day from $days_back till $days_for
if( open( DOFILE, "$dolist" ) ) {
        @do_array = <DOFILE>;
        close (DOFILE);
} else {

        @do_array = "all";
}
if( open( DONEFILE, "$donelist" ) ) {
        @done_array = <DONEFILE>;
        close (DONEFILE);
}
foreach $do_record (@do_array) {
        # Also, we need to add logic to account for blank lines.
        unless ($do_record =~ /^\#/) {
                chomp ($do_record);
                @do_record_list = split(/[\s]+/, $do_record);
                $item_label = $do_record_list[0];
                $hour_phase = 0;
                if (defined ($do_record_list[1])) { $hour_phase = $do_record_list[1]; }
```

```
                $hour_freq = 12;
                if (defined ($do_record_list[2])) { $hour_freq = $do_record_list[2]; }
                if (defined ($do_record_list[3])) { $curr_dtg = $do_record_list[3]; }
                if (defined ($do_record_list[4])) { $days_back = $do_record_list[4]; }
                if (defined ($do_record_list[5])) { $days_for = $do_record_list[5]; }

                # Initialize date-time group variables
                $hours_back = $days_back * -24;
                $init_dtg = timecalc ($curr_dtg, $hours_back);
                $hours_for = $days_for * 24;
                $final_dtg = timecalc ($curr_dtg, $hours_for);
                $dtg = timecalc ($init_dtg, $hour_phase);
                $final_dtg = timecalc ($final_dtg, $hour_phase);
                print "# for record: $do_record: begins $dtg and ends $final_dtg ########################\n";

                # Loop through all date-time groups
                $flag = 0;
                while( $dtg <= $final_dtg && $flag != 2 ) {
                        print "- checking for $dtg $item_label ------------------------------------------------\n";

                        # Initialize completion flag
                        $flag = 0;

                        # Check with all the done list items
                        foreach $done_record (@done_array) {
                                chomp ($done_record);  # removes carriage return
                                @done_item_label = split " ", $done_record ;
                                $count = split " ", $done_record ;
                                if ($done_item_label[$count - 2] eq $dtg && $done_item_label[$count - 1] eq $item_label) {
                                        print "loop_mgr: $program $dtg $item_label already completed\n";
                                        $flag = 1;
                                }
                        }
                        if( $flag == 0 ) {
                                $exit_flag = system( "$program $dtg $item_label" );
                                $date = scalar localtime;
                                chomp ($date);
                                if( $exit_flag == 0 ) {
                                        open (DONEFILE, ">>$donelist");
                                        print DONEFILE "$date $dtg $item_label\n";
                                        close (DONEFILE);
                                        print "loop_mgr: $program $dtg $item_label exited as completed.\n";
                                } else {
                                        open (NOTDONEFILE, ">>$notdonelist");
                                        print NOTDONEFILE "$date $dtg $item_label\n";
                                        close (NOTDONEFILE);
                                        print "loop_mgr: $program $dtg $item_label exited as incomplete -- will be back again
later\n";
                                        if( $lsequence eq "true" ) {
                                                print "loop_mgr: $program $dtg $item_label not completed yet -- cannot go on\n";
                                                $flag = 2;   # means that no more checking for this item
                                        }
                                }
                        }
                        $dtg = timecalc( $dtg, $hour_freq );
                }
        }
        @do_record_list = "";
}

# Limit done-list size
if( open(DONEFILE, $donelist) ) {
        @lines = <DONEFILE>;
        close (DONEFILE);
        if (@lines>50000) {
                $number_to_delete = (@lines-1);
                for ($i=0; $i<$number_to_delete; $i++) {
                        shift (@lines);
                }
                open (TEMPFILE, ">${donelist}.tmp");
                for ($i=0; $i<@lines; $i++) {
                        $line = $lines[$i];
                        print TEMPFILE "$line";
                }
                close (TEMPFILE);
                system ("cp ${donelist}.tmp $donelist");
                unlink ("${donelist}.tmp");
        }
}
exit 0;
```

```
################################################################################
# Authors:
#   - James D. Dykes, NAVO, N212, 23 August 1999
#   - Pat Wilz, PSI
# Revised:
#       19 November 2003
#       15 Sep 06
#               Changed size limit on done-list and delete always only one at a time
#       28 Dec 06
#               Added mail message sent when catching up to curr_dtg
#       05 Oct 07, J Dykes
#               Added $instance environmental variable to simplify specifiying different do-lists, etc.
#       14 May 08, J Dykes
#               Added $lsequence environemnt variable to control whether or not runs can run ahead of sequence.
#               Added additional options to the do-list to define dtg, days_back, and days_for for each item.
#       11 June 08, J Dykes
#               Streamlined control variable definitions with the definevar function

# Purpose:
#   - Using a do- and done-list, runs a program until successful completion
#     for each listed item over a few days

# Method:
#   - Normally used non-interactively

# Usage: loop_mgr <program>
#   - <program>
#     -- depending upon success of completion, should return appropriate exit
#        status for loop_mgr to respond on
#     -- accepts these two arguments in this order
#        --- first arg  - base date time YYYYMMDDHH e.g. 1999092300
#        --- second arg - reference ID

# Control/input/output files:
#   - do-list (edited by user)
#     -- file naming: do_<program>
#     -- contents: reference ID, hour phase, hour interval, base dtg, days back, days forward
#        --- reference ID identifies unique item or task usually a key in a
#            par file used by <program>
#        --- hour phase is the first hour of the day to use for a base
#            date and time (default: 00)
#        --- hour interval e.g. 12 is every 12 hours
#            (default: 12 i.e. twice per day)
#        --- Last three items are the same as defined in the environment variables.
#   - done-list (edited by user and <program>, user delete lines to do again)
#     -- file naming: done_<program>
#     -- contents: results from date command, base date and time, reference ID
#   - notdone-list (output from <program> only)
#     -- file naming: notdone_<program>
#     -- contents: results from date command, base date and time, reference ID

# Environment variables: (user can define to override defaults using the definevar function)
#   - LISTS - directory where do-, done-, and notdone list files are located
#     (default: current dir)
#   - days_back - number of days to go back to start
#   - days_for - number of days to go forward to end
#   - dtg - base date and time i.e. YYYYMMDDHH (default: current date and time
#   - dolist
#   - donelist
#   - notdonelist
#   - instance - as in run_lock used to distiguish processes for lists
#   - lsequence - True if items need to be done in sequence

# Platform Dependencies:
#   - Check the path for perl
#   - Works on everything unix

# Software Dependencies:
#   - $LIB/General.pm contains these perl utilities
#     -- timecalc - adds hours to date and time group returning a new date and
#        time group
#     -- get_date - gets current year, month, day
#     -- definevar - Control variables are defined from the environment
#        or a default is used.

# Bugs/Notes
#   - If only the current date for the base data is required, specify 0.0 vice
#     just 0 for the variable days_back, otherwise it will not get defined
################################################################################
```

Listing of dependencies in $USR/lib/General.pm are as follows:

```
#########################################################
sub get_date
# Gets the current date in this format: YYYYMMDD
# YYYY - year since AD
# MM   - month of the year
# DD   - day of the month
{
   my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst)=
       localtime (time);
   $mon++;
   if ($mon < 10) {
      $mon="0$mon";
   }
   if ($mday < 10) {
      $mday ="0$mday";
   }
   $year+=1900;

   return("$year"."$mon"."$mday");
}

#########################################################
sub definevar
# Control variables are defined from the environment
# or a default is used.
#
# Created: 11 June 2008, J Dykes, NRL-SSC, 7322
{
        local( $controlvar ) = "$_[0]";
        local( $default ) = "$_[1]";

        if( $ENV{"$controlvar"} ) {
                $$controlvar = $ENV{"$controlvar"};
                print "loop_mgr: Using $controlvar = $$controlvar by override\n";
        } else {
                $$controlvar = $default;
                $ENV{"$controlvar"} = $$controlvar;
                print "loop_mgr: Using $controlvar = $$controlvar by default\n";
        }
        return( $$controlvar );
}

#########################################################
sub timecalc                                                    #
# This is a PERL version of the script that will send out #
# a DTG subtracting hours from another DTG. For instance  #
# 97121312 -18 (Dec. 13, 1997 1200 hrs minus 18 hours )   #
# will return 97121218 (Dec 12, 1997 1800 hrs.)           #

{
local $indate="$_[0]";
local $hrchng="$_[1]";
chomp ($indate);
chomp ($hrchng);
local (@dyinmo)=(31,28,31,30,31,30,31,31,30,31,30,31);
my $i;
@indate=split(//,$indate);
if (length($indate) == 8)
{
   $ct="";
   @yr=@indate[0..1];
   $yr=join("",@yr);
   @mo=@indate[2..3];
   $mo=join("",@mo);
   @dy=@indate[4..5];
   $dy=join("",@dy);
   @hr=@indate[6..7];
   $hr=join("",@hr);
}
else
{
   if (length($indate) == 10)
   {
   @ct=@indate[0..1];
   $ct=join("",@ct);
   @yr=@indate[2..3];
   $yr=join("",@yr);
   @mo=@indate[4..5];
   $mo=join("",@mo);
   @dy=@indate[6..7];
```

```
   $dy=join("",@dy);
   @hr=@indate[8..9];
   $hr=join("",@hr);
   }
   else
   {
      return (1);
   }
}
$hr+=$hrchng;
if ($hr>=0 && $hr <=23)
{
# do nothing else
}
else
{
   if ($yr%4 == 0)
      {$dyinmo[1] = 29;}
   $dyofyr = 0;
   if ($mo > 1)
   {
      for ($i = 1; $i < $mo; $i++)
      {
         $dyofyr += $dyinmo[$i-1];
      }
   }
   $dyofyr += $dy;
   while ($hr > 23)
   {
      $hr -= 24;
      $dyofyr++;
   }
   while ($hr < 0)
   {
      $hr += 24;
      $dyofyr--;
   }
   if ($yr%4 == 0)
   {$yrdys = 366;}
   else
      {$yrdys = 365;}
   while ($dyofyr > $yrdys)
   {
      $dyofyr -= $yrdys;
      $yr++;
      if ($yr%4 == 0)
         {$yrdys = 366;}
      else
         {$yrdys = 365;}

   }
   while ($dyofyr < 1)
   {
      if ($yr%4 == 1)
         {$dyofyr += 366;}
      else
         {$dyofyr += 365;}
      $yr--;
   }
   while ($yr > 99)
   {
    $yr -= 100;
    if ( length($indate) == 10 )
       {$ct++;}
   }
   while ($yr < 0)
   {
      $yr += 100;
      if ( length($indate) == 10 )
         {$ct--;}
   }
   if ($yr%4 == 0)
      {$dyinmo[1] = 29;}
   else
      {$dyinmo[1] = 28;}
   $mo = 1;
   while ($dyofyr > $dyinmo[$mo-1])
   {
      $dyofyr -= $dyinmo[$mo-1];
      $mo++;
   }
   $dy=$dyofyr;
}
```

```
#$newdate=$hr+100*($dy+100*($mo+100*($yr+100*$ct)));
 if (length($yr) < 2)
 {
     $yr="0".$yr;
 }
 if (length($mo) < 2 )
 {
     $mo="0".$mo;
 }
 if (length($dy) < 2)
 {
     $dy="0".$dy;
 }
 if (length($hr) < 2)
 {
    $hr ="0".$hr;
 }
 $newdate=$ct.$yr.$mo.$dy.$hr;
return ($newdate);
}
```

A.3 Run Locking

If an application is running and you do not want a duplicate instance of that application running, then invoke this utility, *run_lock.pl* (linked to *run_lock*), which is especially useful if an instance of an application is kicked off automatically and frequently. Figure 2 shows the implementation of *run_lock*.

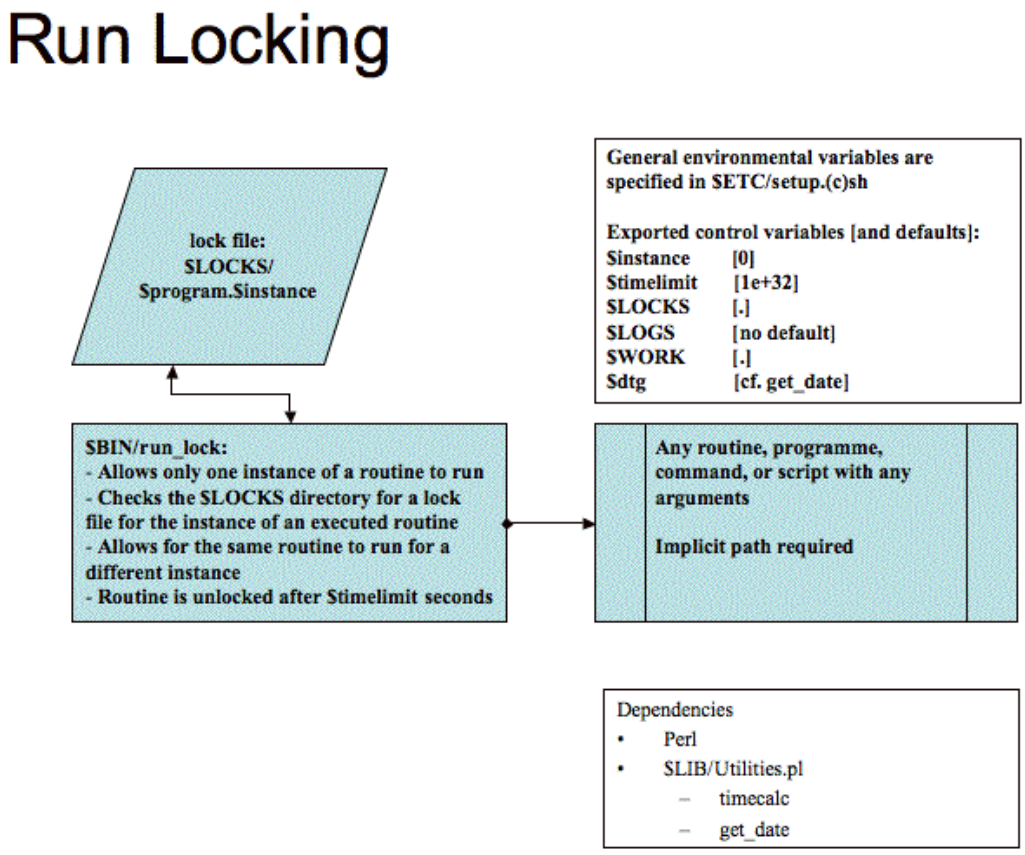

Figure 2 Diagram showing implementation of run locking procedure.

Sample command sequence to run the *ls* command, have the lock file in *$LOCKS* and use an instance name other than the default, 0:

```
export instance ls_special
run_lock ls $LOCKS 1>$LOGS/log-$instance 2>&1 &
```

A lock file called, *ls.ls_special* is written in *$LOCKS*, though very briefly since this command does not take long, but then the files in $LOGS would show that file listed. The *run_lock* utility is useful to check for an application that should be running constantly, such that should a failure kill that application, it will come back up when scheduled in the crontab.

Listing of the run_lock script is as follows:

```
#!/usr/bin/perl
# See documentation below

use Env;
require "$USR/lib/General.pm";

$SIG{INT} = 'handler';
$SIG{HUP} = 'handler';
$SIG{QUIT} = 'handler';
$SIG{XCPU} = 'handler';

# Command to lock
$command = $ARGV[0];
if ( ! ($command = $ARGV[0])) {
        print "run_lock: What? no command?\n";
        exit;
}
print "run_lock: Running $command\n";

$LOCKS = definevar( "LOCKS", "." );
$instance = definevar( "instance", 0 );
$lockfile = "$LOCKS/${command}.${instance}";
$LOGS = definevar( "LOGS", "." );
$logfile = "$LOGS/${command}.${instance}";
$timelimit = definevar( "timelimit", 1e+32 );

# Lock file check
if (-s $lockfile) {
        open (INFILE,$lockfile);
        $PID = <INFILE>;
        chomp ($PID);
        close (INFILE);
        $pstatus = `ps -p $PID | grep $PID`;
        if ($pstatus ne "") {
                # Lock file age
                $ctime = (stat ($lockfile))[10];
                $epochtime = time ();
                $timediff = $epochtime - $ctime;
                if ($timediff > $timelimit) {
                        print "run_lock: lock file exceeds time limit $timelimit secs\n";
                        print "run_lock: killing hung PID $PID\n";
                        system ("kill -9 $PID");
                        unlink "$lockfile";
                } else {
                        print "run_lock: lock file exists within time limit - exiting\n";
                        exit;
                }
        } else {
                unlink "$lockfile";
        }
}

open (OUTFILE,">$lockfile");
$PID = $$;
print OUTFILE "$PID\n";
print OUTFILE "timelimit = $timelimit\n";
close (OUTFILE);

#  Establish work directory if desired
if ($ENV{"WORK"}) {
        $WORK = $ENV{"WORK"};
```

```
        $TMPWORK = "$WORK/${command}.${PID}";
        print "run_lock: Using TMPWORK = $TMPWORK\n";
        mkdir $TMPWORK,0755;
        chdir $TMPWORK;
} else {
        $WORK = ".";
        $TMPWORK = "$WORK";
}
$date2 = `date`;
print "== BEGIN PID $PID $date2\n";
system( "echo == BEGIN PID $PID $date2 1>$logfile 2>&1" );

# Execute command with arguments
if ($ENV{"LOGS"}) {
        system ("@ARGV 1>$logfile 2>&1");
} else {
        system ("@ARGV");
}

$date2 = `date`;
print "== END PID $PID $date2\n";
unlink "$lockfile";
if ($ENV{"WORK"}) {
        chdir "$TMPWORK";
        chdir "..";
        print "run_lock: Removing TMPWORK = $TMPWORK\n";
        system ("rm -rf $TMPWORK");
}
exit 0;

################################################################################
# Authors:
#   - James D. Dykes, NAVO, N212, 13 August 1999
#   - Pat Wilz, PSI
# modified
#       05 Oct 07
#               Mostly streamlining code, eliminated dtg references
#       11 Jun 08, J Dykes
#               Streamlined control variable definitions with the definevar function

# Purpose:
#   - Avoids duplicate runs of a command doing the same task

# Method:
#   - Duplicate runs of a command can run for different tasks, which are made
#     unique by $instance
#   - Uses a temporary work directory if so desired as controlled by $WORK
#   - Processes that exceed $timelimit are killed allowing a new task may start
#   - Normally used non-interactively

# Environment varibles: (user can define to override defaults)
#   - LOCKS - directory where lock file is located (default: current dir)
#   - LOGS - directory of file to where STDOUT is directed (default: to stdout)
#   - WORK  - directory where work directory is created (default: current dir)
#   - instance - unique value to distinguish between multiple instances of the
#     same command (default: first argument of command, otherwise 0)

# Platform Dependencies:
#   - Check the path for perl
#   - A little quirky at line 5, but had to accomodate Cray's problem testing
#     undefined variables.  Still works on everything else unix

# Software Dependencies: Perl 5
#   - definevar in Utilities.pm (General.pm) in $USR/lib

# Bugs/Notes
#   - The $program variable needs to be only the program with no path on it, so
#     you will have to put it in the path to execute it as a name alone.
################################################################################
```

The one dependency has already been listed in the above section.

A.4 Implementation

Figure 3 illustrates the flow of the general implementation using a *cronpanel* script to keep things going 365/24/7.

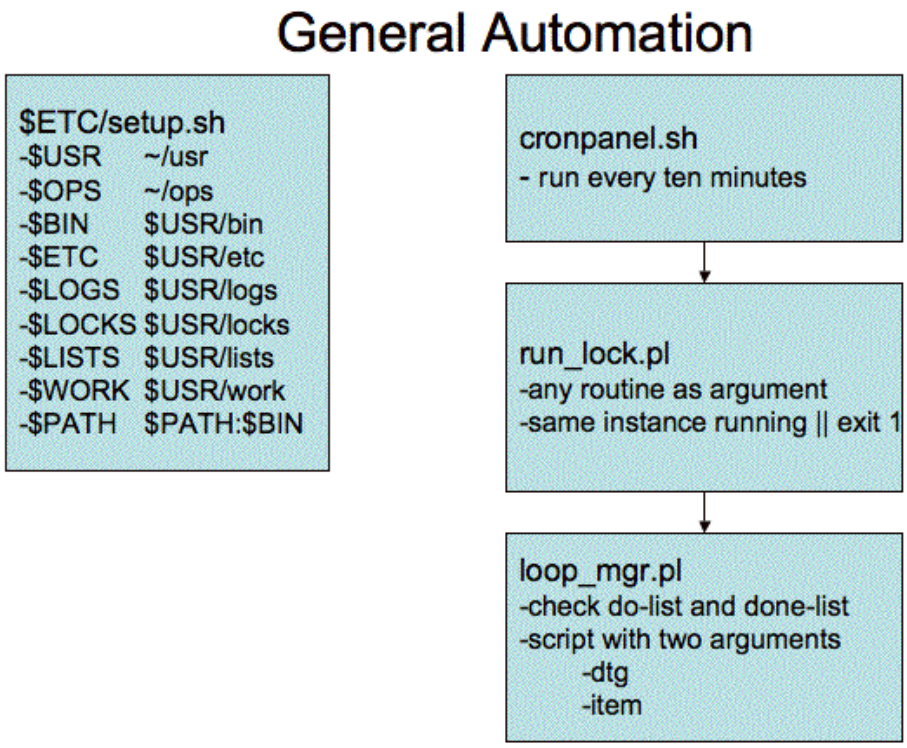

Figure 3 Layout of overall automation system.

The main *cronpanel* script in *$BIN* is kicked off by a crontab and will kick off other *cronpanel* scripts within their various branches of applications. Together the two Perl scripts work as a general automation team.

Here is simple script snippet used in the context of the automation system using *loop_mgr*. The first argument is always the 10-digit date, *$dtg*, YYYYMMDDYY. The second argument is the *$item* which tries to include all unique aspects specific to this *$item*. This item is listed in the do-list as *$LISTS/do_$instance*. The second argument could be parsed into sub-arguments as needed in the script below where things like TAU are needed to run a unique task. Strategically located exit status points tell *loop_mgr* of completion (status 0) or non-completion (status other than 0). Upon completion, this *$item* and *$dtg* are listed in *$LISTS/done_$instance*.

```
dtg=$1
item=$2   #  e.g. ncom_relo_anex32:0:3:72
nc_root=`echo $item | cut -f1 -d:`
itau=`echo $item | cut -f2 -d:`
tau_inc=`echo $item | cut -f3 -d:`
mtau=`echo $item | cut -f4 -d:`
do certain things || exit 1  # way to exit if it doesn't work out
etc. ...
exit 0   #  successful completion
```

The *run_lock* command can be used to keep only one instance of *loop_mgr* running at any one time.

Appendix B – AutoMetrics Scripts

The three scripts listed here play the pivotal roles in AutoMetrics as described in this document. The following is a listing of the *matchup_set* script:

```
#!/bin/ksh -x

# Purpose: Builds files that match observations to forecasts from NCOM and HYCOM
#       Customised for NAVO forecast ops and configured to be run with loop_mgr.
# revised 04 Nov 10, J Dykes, NRL-SSC, 7322
# modified:
#       21 Feb 11: J Dykes: persistence work is removed.
#       06 Apr 11: Options for multiple architectures are included.

dtg=$1
item=$2   #  e.g. ncom_relo_anex32:0:3:72

[ -d $ARTP_OPS/run/$item ] || mkdir -p $ARTP_OPS/run/$item
cd $ARTP_OPS/run/$item

if [ -f jobdone-$dtg ]; then
        rm jobdone-$dtg
        exit 0
fi

[ -f jobready ] && exit 1
if [ -f jobrunning ]; then
        job_id=`cat jobrunning`
        qstat | grep $job_id
        if [ $? -eq 0 ]; then
                echo In case $case: a job still running
                exit 1
        else
                rm jobrunning
        fi
fi
[ -f jobpostprocessing ] && exit 1

nc_root=`echo $item | cut -f1 -d:`
ob_type=profile
itau=`echo $item | cut -f2 -d:`
tau_inc=`echo $item | cut -f3 -d:`
mtau=`echo $item | cut -f4 -d:`

yyyy=`echo $dtg | cut -c1-4`
mm=`echo $dtg | cut -c5-6`
dd=`echo $dtg | cut -c6-7`

matchprefix=daily_${itau}_${mtau}_${ob_type}_${nc_root}
[ -f  $matchprefix-$dtg-interp.matchup ] && rm $matchprefix-$dtg-interp.matchup

rm list_obs_daily_${itau}_${mtau}_${ob_type}
touch list_obs_daily_${itau}_${mtau}_${ob_type}
rm list_ncfiles
touch list_ncfiles

#### Gather the appropriate obs files (always starts at 00Z)
idtg=${yyyy}${mm}${dd}00
mdtg=`timecalc $dtg $mtau`
while [ $idtg -le $mdtg ]; do
        for dist in navoqc_public navoqc_rstrct
        do
                OOC_PROFILE_DIR=/u/home/ooc/models/ncoda/etc/$dist/$ob_type
                #OOC_PROFILE_DIR=/scr/ooc/models/ARTP/data/ocnqc/$dist/$ob_type
                MY_PROFILE_DIR=$ARTP_OPS/data/ocnqc/$dist/$ob_type
                [ -d $MY_PROFILE_DIR ] || mkdir -p $MY_PROFILE_DIR
                if [ -f $OOC_PROFILE_DIR/$idtg.$ob_type ]; then
                        cp $OOC_PROFILE_DIR/$idtg.$ob_type $MY_PROFILE_DIR
                        chmod 660 $MY_PROFILE_DIR/$idtg.$ob_type
                        chgrp NAVOSOOC $MY_PROFILE_DIR/$idtg.$ob_type
                        #rcp -p $MY_PROFILE_DIR/$idtg.$ob_type
${ARCHIVE_HOST}:${ARCHIVE_HOME}/models/ARTP/data/ocnqc/$dist/$ob_type
                        #rsh ${ARCHIVE_HOST} chgrp NAVOSOOC ${ARCHIVE_HOME}/models/ARTP/data/ocnqc/$dist/$ob_type/$idtg.$ob_type
                else
                        [ -f $MY_PROFILE_DIR/$idtg.$ob_type ] || rcp
${ARCHIVE_HOST}:${ARCHIVE_HOME}/models/ARTP/data/ocnqc/$dist/$ob_type/$idtg.$ob_type $MY_PROFILE_DIR
                fi
                [ -f $MY_PROFILE_DIR/$idtg.$ob_type ] || exit 1
                echo $MY_PROFILE_DIR/$idtg.$ob_type >> list_obs_daily_${itau}_${mtau}_${ob_type}
```

```
        done
        idtg=`timecalc $idtg 24`

        if [ ! -s list_obs_daily_${itau}_${mtau}_${ob_type} ]; then
                echo "matchup: No obs to match up with at this time."
                exit 1
        fi
done
#### Process the appropriate netCDF model files
OOC_NETCDF_DIR=/scr/ooc/data/ncom/packed_coards
tau=$itau
while [ $tau -lt $mtau ]; do
        tau=`add_digits $tau 3`
        tau_next=`echo $tau + $tau_inc | bc`
        tau_next=`add_digits $tau_next 3`

        MY_NETCDF_DIR=$ARTP_OPS/data/netCDF
        [ -d $MY_NETCDF_DIR ] || mkdir -p $MY_NETCDF_DIR
        domain_ncom_relo=`echo $nc_root | grep ncom_relo | cut -c11-72`
        domain_ncom_glb=`echo $nc_root | grep ncom_glb | cut -c10-72`

        # Section for NCOM/HYCOM runs if applicable
        if [ ! -f ${nc_root}_${dtg}_t${tau}-t${tau_next}.nc ]; then
                if [ -s ${OOC_NETCDF_DIR}/${nc_root}_${dtg}_t${tau}.nc ] && [ -s
${OOC_NETCDF_DIR}/${nc_root}_${dtg}_t${tau_next}.nc ]; then
                        ncrcat -O ${OOC_NETCDF_DIR}/${nc_root}_${dtg}_t${tau}.nc
${OOC_NETCDF_DIR}/${nc_root}_${dtg}_t${tau_next}.nc ${nc_root}_${dtg}_t${tau}-t${tau_next}.nc
                        echo ${nc_root}_${dtg}_t${tau}-t${tau_next}.nc >> list_ncfiles
                        echo matchup_set: Accessing netCDF files from OOC and catting pairs.
                elif [ -s ${MY_NETCDF_DIR}/${nc_root}_${dtg}_t${tau}.nc ] && [ -s
${MY_NETCDF_DIR}/${nc_root}_${dtg}_t${tau_next}.nc ]; then
                        ncrcat -O ${MY_NETCDF_DIR}/${nc_root}_${dtg}_t${tau}.nc
${MY_NETCDF_DIR}/${nc_root}_${dtg}_t${tau_next}.nc ${nc_root}_${dtg}_t${tau}-t${tau_next}.nc
                        echo ${nc_root}_${dtg}_t${tau}-t${tau_next}.nc >> list_ncfiles
                        echo matchup_set: Accessing netCDF files from local directory and catting pairs.
                elif [ $domain_ncom_relo ]; then
                        #### NCOM Regional   # e.g. ncom_relo_amseas_2010100100.tar.gz
                        if [ ! -f $MY_NETCDF_DIR/${nc_root}_${dtg}.tar ]; then
                                rcp $ARCHIVE_HOST:$ARCHIVE_OOC/relo/$domain_ncom_relo/Nc/${nc_root}_${dtg}.tar.gz $MY_NETCDF_DIR
                                [ -f $MY_NETCDF_DIR/${nc_root}_${dtg}.tar.gz ] || rcp
$ARCHIVE_HOST:$ARCHIVE_OOC/relo/$domain_ncom_relo/Nc/${yyyy}${mm}/${nc_root}_${dtg}.tar.gz $MY_NETCDF_DIR
                                gzip -d $MY_NETCDF_DIR/${nc_root}_${dtg}.tar.gz || exit 1
                        fi
                        tar xf $MY_NETCDF_DIR/${nc_root}_${dtg}.tar || exit 1
                        mv ${nc_root}_${dtg}_t*.nc $MY_NETCDF_DIR
                        ncrcat -O ${MY_NETCDF_DIR}/${nc_root}_${dtg}_t${tau}.nc
${MY_NETCDF_DIR}/${nc_root}_${dtg}_t${tau_next}.nc ${nc_root}_${dtg}_t${tau}-t${tau_next}.nc || exit 1
                        echo ${nc_root}_${dtg}_t${tau}-t${tau_next}.nc >> list_ncfiles
                        echo matchup_set: Accessing NCOM regional netCDF files from archive directory and catting pairs.
                elif [ $domain_ncom_glb ]; then
                        #### NCOM Global cut-outs   # e.g. ncom_glb_regp01_2010100100.nc.gz
                        if [ ! -f $MY_NETCDF_DIR/${nc_root}_${dtg}.nc ]; then
                                rcp $ARCHIVE_HOST:$ARCHIVE_OOC/data/ncom1/glb8_3b/work/$domain_ncom_glb/${nc_root}_${dtg}.nc.gz
$MY_NETCDF_DIR
                                [ -f $MY_NETCDF_DIR/${nc_root}_${dtg}.nc.gz ] || rcp
$ARCHIVE_HOST:$ARCHIVE_OOC/data/ncom1/glb8_3b/work/$domain_ncom_glb/${yyyy}${mm}/${nc_root}_${dtg}.nc.gz $MY_NETCDF_DIR
                                gzip -d $MY_NETCDF_DIR/${nc_root}_${dtg}.nc.gz || exit 1
                        fi
                        echo $MY_NETCDF_DIR/${nc_root}_${dtg}.nc > list_ncfiles
                        echo matchup_set: Accessing NCOM global netCDF files from archive directory
                else
                        echo matchup_set: cannot make file: ${nc_root}_${dtg}_t${tau}-t${tau_next}.nc -- skipping ...
                fi
        else
                echo ${nc_root}_${dtg}_t${tau}-t${tau_next}.nc >> list_ncfiles || exit 1
                echo matchup_set: Target netCDF files already in local run directory
        fi

        tau=$tau_next
        idtg=`timecalc $dtg $tau`
done

[ -s list_ncfiles ] || exit 1

#### Set up for submission into PBS
echo $dtg > submitdtg
echo daily_${itau}_${mtau}_${ob_type} > submitcategory
echo $nc_root > submitnc_root

arch=`uname -a | awk '{print $1}'`
if [ $arch == "Linux" ]; then
        options="-q standard -l mppwidth=1 -l mppnppn=1"
elif [ $arch == "AIX" ]; then
```

```
        options="-q share -l select=1"
else
        echo no valid machine architecture -- exiting.
        exit 1
fi
qsub $options $ARTP_HOME/bin/matchup_agent.job 1>jobready 2>&1
exit 1
```

The following is a listing of the *matchup_agent.job* script:

```
#!/bin/ksh -x

# Submitted script which runs the AutoMetrics programme, matchup_drvr.x in PBS

# created
#       2 Mar 11, J Dykes, NRL-SSC, 7320
# modified:
#       14 Mar 11, J Dykes
#       - changed arguments for the newer version of matchup.x programme
#       06 Apr 11
#       - Removed '|| exit 1' portion from the aprun and poe run lines to keep the system from stopping here.
#       - Changed archiving to simply tar cf all the monthly matchup files each time.
#       - Architecture-dependent options removed and chosen by the matchup_set script.
#       17 Apr 11, changed the way tar files are handled to eliminate duplicative entries.

#PBS -o log-ARTP
#PBS -e log-ARTP
#PBS -A NRLSS03745060
##PBS -A NAVOSOOC
#PBS -l walltime=2:30:00
#PBS -q share
##PBS -q internal
#PBS -m ae
#PBS -N AutoMetrics
#PBS -l application=autometrics

# Other needed predifined (usually by batch queuing system)
#  $PBS_O_WORKDIR
#  $PBS_JOBID

cd $PBS_O_WORKDIR

umask 002
echo $PBS_JOBID 1> jobrunning 2>&1
rm jobready

. ~/usr/etc/setup.sh
. ~/models/ARTP/etc/setup.sh

dtg=`cat submitdtg`
category=`cat submitcategory`
nc_root=`cat submitnc_root`
matchprefix=${category}_${nc_root}
npes=1
nslots=1

arch=`uname -a | awk '{print $1}'`
if [ $arch == "Linux" ]; then
        aprun -N $npes -n $nslots $ARTP_HOME/bin/matchup_drvr.x list_ncfiles list_obs_${category} ${matchprefix}-${dtg}-
interp.matchup 1>log-matchup-${dtg}-${matchprefix}-interp-$$ 2>&1
elif [ $arch == "AIX" ]; then
        poe $ARTP_HOME/bin/matchup_drvr.x list_ncfiles list_obs_${category} ${matchprefix}-${dtg}-interp.matchup 1>log-matchup-
${dtg}-${matchprefix}-interp-$$ 2>&1
else
        echo no valid architecture for PBS submission -- exiting.
        exit 1
fi

[ $? -eq 0 ] || exit $?

mv jobrunning jobpostprocessing

#### Archive the results
touch $matchprefix-$dtg-interp.matchup
filesize1=`filesize $matchprefix-${dtg}-interp.matchup`
if [ $filesize1 -gt 4 ]; then
        MATCHUPS=$ARTP_OPS/data/matchups
        TARFILES=$ARTP_OPS/data/tarfiles
        [ -d $MATCHUPS ] || mkdir -p $MATCHUPS
        [ -d $TARFILES ] || mkdir -p $TARFILES
```

```
        cd $MATCHUPS

        yyyy=`echo $dtg | cut -c1-4`
        mm=`echo $dtg | cut -c5-6`

        rcp -p ${ARCHIVE_HOST}:${ARCHIVE_HOME}/models/ARTP/data/tarfiles/${matchprefix}-${yyyy}${mm}-interp.matchup.tar $TARFILES
        if [ -f $TARFILES/${matchprefix}-${yyyy}${mm}-interp.matchup.tar ]; then
                tar xf $TARFILES/${matchprefix}-${yyyy}${mm}-interp.matchup.tar
        fi
        cp $PBS_O_WORKDIR/${matchprefix}-${dtg}-interp.matchup . || exit 1
        tar cf $TARFILES/${matchprefix}-${yyyy}${mm}-interp.matchup.tar ${matchprefix}-${yyyy}${mm}*-interp.matchup || exit 1
        chmod 660 ${TARFILES}/${matchprefix}-${yyyy}${mm}-interp.matchup.tar
        chgrp NAVOSOOC ${TARFILES}/${matchprefix}-${yyyy}${mm}-interp.matchup.tar
        rcp -p $TARFILES/${matchprefix}-${yyyy}${mm}-interp.matchup.tar ${ARCHIVE_HOST}:${ARCHIVE_HOME}/models/ARTP/data/tarfiles
|| exit 1
        rsh ${ARCHIVE_HOST} chgrp NAVOSOOC ${ARCHIVE_HOME}/models/ARTP/data/tarfiles/${matchprefix}-${yyyy}${mm}-
interp.matchup.tar
        cd $PBS_O_WORKDIR
fi

mv jobpostprocessing jobdone-$dtg
rm sub* *.matchup *.nc
exit 0
```

The following is a listing of the *ncdiffr* script:

```
#!/bin/sh -x

# Purpose: Produce differences between two NCOM model fields of all variables
# Issues: It is assumed that each TAU is in separate files
# created 15 Sep 09, J Dykes, NRL-SSC, 7322
# modified:
#       22 Sep 09, simplified and adapted for more genaralised netCDF model output, like COAMPS

dtg=$1
item=$2  # e.g. ncom_relo_oksw16:24:48, so that TAU 48 from forecast before is subtracted from TAU 24 of this $dtg

nc_root=`echo $item | cut -f1 -d:`
currenttau=`echo $item | cut -f2 -d:`
beforetau=`echo $item | cut -f3 -d:`
currenttau=`add_digits $currenttau 3`
beforetau=`add_digits $beforetau 3`

#  Example nc_root
# ncom_glb_ecse
# ncom_relo_oknwtrgh_v4
# coamps_ecs

#### Gather the appropriate netCDF model files
NETCDF_DIR=$ARTP_OPS/data/netCDF/model-model
[ -d $NETCDF_DIR ] || mkdir -p $NETCDF_DIR
cd $NETCDF_DIR
for wtau in $currenttau $beforetau
do
        tauback=`echo $wtau - $currenttau | bc`
        wdtg=`timecalc $dtg -$tauback`
        ncfile=${nc_root}_${wdtg}_t${wtau}.nc

        # Looking for files in the local collection point
        if [ ! -f $ncfile ]; then

                # Checking for any any model output files not broken down into individual files yet
                if [ -f ../netCDF/hold/${nc_root}_${wdtg}.nc ]; then
                        ln ../netCDF/hold/${nc_root}_${wdtg}.nc .
                        ncks -d time,$wtau,$wtau ${nc_root}_${wdtg}.nc $ncfile
                fi

                # Checking for NCOM relo files
                nc_part1=`echo $nc_root | cut -c1-9`
                if [ $nc_part1 == "ncom_relo" ]; then

                        # Checking for NCOM relo files on local real-time directory
                        if [ -f /scr/ooc/data/ncom/packed_coards/$ncfile ]; then
                                ln /scr/ooc/data/ncom/packed_coards/$ncfile .
                        fi

                        # Checking for NCOM relo tar files in archive
                        domain=`echo $nc_root | cut -c11-72`
                        if [ ! -f ${nc_root}_${wdtg}.tar ]; then
                                wyyyy=`echo $wdtg | cut -c1-4`
                                wmm=`echo $wdtg | cut -c5-6`
```

```
                                rcp $ARCHIVE_MACH:$ARCHIVE_OOC/relo/$domain/Nc/${nc_root}_${wdtg}.tar.gz .
                                [ -f ${nc_root}_${wdtg}.tar.gz ] || rcp
$ARCHIVE_MACH:$ARCHIVE_OOC/relo/$domain/Nc/${wyyyy}${wmm}/${nc_root}_${wdtg}.tar.gz .
                                [ -f ${nc_root}_${wdtg}.tar.gz ] && gzip -d ${nc_root}_${wdtg}.tar.gz
                                [ -f ${nc_root}_${wdtg}.tar.gz ] && rm ${nc_root}_${wdtg}.tar.gz
                        fi
                        if [ -f ${nc_root}_${wdtg}.tar ]; then
                                tar xf ${nc_root}_${wdtg}.tar
                                rm ${nc_root}_${wdtg}.tar
                        fi
                fi
        fi
        if [ ! -e $ncfile ]; then
                rm ${nc_root}_${dtg}*.nc
                exit 1
        fi
done

tauback=`echo $beforetau - $currenttau | bc`
beforedtg=`timecalc $dtg -$tauback`
nowfile=${nc_root}_${dtg}_t${currenttau}.nc
beforefile=${nc_root}_${beforedtg}_t${beforetau}.nc

difffile=${nc_root}_${dtg}_t${currenttau}-t${beforetau}.nc
ncdiff -y sbt -x -v tau -O $nowfile $beforefile $difffile

if [ $? == 0 ]; then
        NCCOMP=$ARTP_OPS/data/model-model
        [ -d $NCCOMP ] || mkdir -p $NCCOMP
        mv $difffile $NCCOMP
        if [ $? != 0 ]; then
                rm ${nc_root}_${dtg}*.nc
                exit 1
        fi
else
        rm $difffile
fi

#### Clean 'er up
rm ${nc_root}_${beforedtg}*.nc ${nc_root}_${dtg}*.nc
find $NCCOMP -type f -mtime +7 -exec rm {} \;
exit 0
```