NEONS at the Finnish Meteorological Institute

The paper to document the speech given by
Veikko Nyfors
At the 5.th Workshop on
Meteorological Operational Systems
14.11.1995 at ECMWF

A project called WETO (for WEather service TOols) has been going on for some time now at FMI. The aim of the project is to reorganize the weather forecasting production environment into more structured and modern form. Foil 1 describes the the plan how the new environment should go in general level.

My colleague Juha Kilpinen will present this WETO project in more general terms later on this week. I will be concentrating in my presentation on one of the very hearts of this system, the database. And even more specifically, in the the short term, transient portion of the database.

Some people call the short term, transient database the 'real-time' database, but that is in my opinion a bit misleading, as we are really not trying to reach the 'millisecond' level of the general real-time aspect. Instead, we are talking about a database, that can support the meteorologist working in an on-line customer situation fast enough with up-to-date data in a reliable and flexible manner.

I would define the characteristic requirements for the transient database as follows:

1) The data should be available in the transient database as soon as possible. In practice, we are talking about a few minutes after the time the data has been produced or received thatm it should be available for the users of the database.

2) The data access to the transient database should be fast. In practice we are talking about seconds to receive some 500 observations or a few GRIDs.

3) The data doesn't have to be stored for ages, but typically for a few days, perhaps a week. The transient database will be used in analysing the current or the very nearby past or future situations. Climatological database, which has totally different caharacteristic requirements, should be used for historical research and such. This makes the implementation of the transient database easier from the performance point of view and also in some other respects.

The traditional way to implement a transient database has been on filesystem based directory tree structures. In the past it really has been the only way to implement a fast enough system. This kind of implementation has problems with providing the data to different platforms, which easily leads to the situation, where the same data is in multiple places in a bit different formats. Also the applications tend to have a lot of coding directly depending on the particular implementation, making them very dependent on environment.

From the beginning of 90's, efficient and cost efficient enough RDBMS systems have been available to implement a transient meteorological data storage. With RDBMS one can easily establish one central data store which everybody is using. The fastest straightforward way to implement a transient met data storage this way unfortunately leaves the problem of the applications still having coding, which is implementation dependent (e.g. table structures of actual data).

RDBMS gives a lot of flexibility for the implementation. One can implement a dynamic environment, where the first thing stored in the database is the definition of the characteristics of the primary data itself to be stored in the database. To access the primary data, the access routines have first to dig out the characteristics of the primary data from the descriptive/associative tables and then to retrieve the primary data based on this info. This

means, that the meteorological information can change in format without raising a need to rewrite the access routines.

To design and implement this kind of a dynamic RDBMS system is a major task. It is essential, that the metadata (descriptive/associative) design is flexible enough to be sufficient well into the future. Fortunately, there are already such dynamic implementations available. One such is the Naval Environmental and Oceanographic Nowcasting System, developed in NRL.

At FMI, we started to evaluate the suitability of the NEONS NRL version in our environment. We found out the idea of dynamic presentation of the data very useful. But we also found out, that further development would still have to be carried out in the design, exspecially in the point data for observations and forecasts. Another area of big effort would have been the creation of the operational environment: automating the loading of data, cycling the tables, cleaning up and so forth.

We had good luck to get in contact with Meteo-France in Toulouse, where a project to do additional development on top of the NRL NEONS had been going on for some time already. The people at Meteo-France were very kind to give us a copy of their operational NEONS-Meteof system for evaluation. We went on with evaluation based on the NEONS-Meteof.

In the rest of the presentation I will first bring out the highlights of the NEONS-Meteof system. Next I will describe the effort to port the NEONS-Meteof system to our heterogeneous evaluation platform. Following this I will present the data flow how to load the data into NEONS and on the other side how to retrieve data from NEONS. Finally I will say a few words on the planned near future developments.

1. NEONS-Meteof highlights

We have the RDBMS of Oracle 7 in wide use at our site as a climatological database implementation and also otherwise. It was very natural for us to prefer the NEONS-Meteof system, which was already ported on the Oracle 7 RDBMS kernel.
Into NEONS-Meteof there has been integrated a sophisticated constraint scheme, where dependencies between metadata tables and other general restrictions have been defined. This aids up a lot in defining the meteorological datatypes to be processed through the database. Without the constraints, it would be too easy for a novice administrator to define characteristics of data in insufficient or even in conflicting way, which would lead into problems later on when accessing the data.
An efficient index hierarchy has been designed and will be automatically created for the data in database. To implement effective indexing from scratch, one has to spend quite some time in analysing the structures of the tables and the general use of data. Without proper indexing, the RDBMS based systems may end up in being real slow.
To speed up the access to the data, a cursor based 'blocked' data retrieve scheme has been implemented in NEONS-Meteof. This minimizes the overhead of database access, as a block of lines (50, parameterizable at build time) is retrieved by one database read.
One has the flexibility of defining basic sql clauses (where, order by, etc...) in the c interface routines, without writing a line of proc code. This is very handy, but may slow down the performance, as it may make the use of the indexes impossible for the RDBMS system. One should be very carefull if using this feature widely in production environment.
In general, I have the feeling, that NEONS-Meteof is making a very good use of Oracle features. On the other hand, this might also bring a bit of Oracle dependency in the implementation.

The logon sceme to the database has been designed so, that only 3 named user licenses are required for the NEONS-Meteof db access. This reduces the costs of the RDBMS licenses significantly at least in Finland.
Two named users are used for general administration and to insert the primary data into the database. All the end users use the third named user to read access the data in the database.

The connecting to the database is made transparent to the end user by a sophisticated logic based on environmental variables. All the end user has to say is 'setupneons' and the NEONS-Meteof database is available for use.

On the administrational and operational area a lot of additional functionality has been developed on NEONS-Meteof.
The cycling of the transient data tables has been fully automated. The information of what kind of tables are required for the next day, is stored in the database's descriptive tables. Thus the definitions of the tables can be modified dynamically on the fly, as the requirements change.
The loading of data to the database has been totally automated in a manner, that suits for a 'real-time' system very well. All one has to worry about, is to make the data to be loaded available in a file (GRIB, BUFR format) in a directory defined by appropriate environmental variable.
Event logging for numerous events in the operational environment has been incorporated in the system.

In NEONS-Meteof there is a scheme for archiving the primary data after the transient part of its lifecycle. The descriptive and associative data can be left on store in the NEONS-Meteof tables, even if the primary data has been stored somewhere else on more cost-effective but slower devices. The pointers to the data are switched from pointing to the 'on-line' primary data tables of the NEONS-Meteof system to point to the place where the data is archived for longer periods. This facility provides a nice platform to combine the transient database and the climatological database into one combined database; end users are not interested whether the data is coming from the transient or the climatological database.

For observations, there is the llt_bufr datatype.
This datatype has a possibility to store up to 25 parameters in float format. The access is fast to these float parameters. A lot of applications prefer to get float parameters directly from the database. The most frequently used float parameters can be stored in these columns.
Observation is also stored as a whole in BUFR format. This makes the life easier for applications that eat the BUFR format directly. Actually one can store the observations in any bitstream format, the format is dynamically specified for each observation type. The next common format to BUFR assumably would be straight ASCII.
One is able to store multiple versions of one specific observation. The initial 'raw' observation that is loaded to the database asap, will be updated with new corrective versions when additional validity checking has been performed. The data retriever has access to any of these versions as wished.

Point forecasts are stored with previ datatype.
With this datatype one can handle up to 50 float parameters per point forecast type. No bitsream type data can be stored. This previ datatype has some additional information specific to forecasts in the metadata definition if compared to the lltbufr, such as the producer of the forecast and the validity of the forecast.

Gridded forecasts are stored as grid datatype.
Data access to the grid datatype is strictly in GRIB format.

Flag datatype is designed for quality control of observations. We haven't had the time yet to make an evaluation of this datatype.

2. NEONS-Meteof port at FMI

The platform on which the NEONS is running at Toulouse is purely HP/HPUX and Oracle 7. Our evaluation environment is much more heterogeneous. The Database Cruncher itself is a Dec/Alpha with DigitalUnix and Oracle 7. The NEONS load server processes are run on a SGI/IRIX system over sqlnet. The Clients at the moment are SGI/IRIX and PC/WNT platforms via sqlnet.

To port the NEONS-Meteof to this heterogeneous evaluation platform of ours required some modifications of syntactical nature, a few logical fixes and even a couple of functional changes.

On syntactical area there were some Makefile mods to be done. A problem with IRIX proc/CPP interaction had to be fixed on Makefile level.
The IRIX proc version in use at FMI had some differences in allowing long concatenated lines. Also some other minor syntactical differences required attention.
In a few places some non-dynamic string arrays had to be increased, as they overflew in our environment due to the basic pathnames being much longer than in Toulouse.

To import the basic database table hierarchy into our database cruncher was relatively easy.

Altogether the effort up to this point to have the NEONS-Meteof available for further evaluation on our environment was close to trivial. The actions required were the typical issues one has to consider when doing cross platform unix ports.

At Meteo-France the files to be loaded to the database are made available with their local 'Transfer Agent' system. We don't have such a system available, so this part had to be implemented in a different way. Our implementation is simply based on files appearing into load directories defined by environmental variables. There is logic involved to prevent files from being multiply loaded in problem situations, to save loaded files aside for a while for analysis if required, the failed files are saved for analysis of the failure and so on. The same one function to get the next file to be loaded into NEONS is used for lltbufr and previ datatypes. The grid datatype is going to be modified to make use of this same function as well, so we would have a standard way to find the loadable files with all the datatypes we use.

The monitoring at Toulouse is based on their local monitor system. We had to rip the monitoring functionality off, as our SMS monitoring system is not usable straight away in the daemon based philosophy of the neons load services. We have in plans to do minor modifications in the philosophy where ever possible to make the processes more batch oriented and thus suitable for SMS control. For the time being we are running mainly on logfile and mail basis.

One of the biggest tasks was to prepare the observation data in proper format suitable for loading into NEONS. The ASCII messages from GTS had to be converted to BUFR preceded with some associative info plus the float column parameter values. The ECMWF code package available for these purposes was used as the basis.
The GRID datatype was easy in this sense, as the file to be loaded into NEONS at FMI is a pure GRIB file in exactly the format as the original producer has created it.

One time consuming phase was to define the first appeareances of each datatype. The documentation is not on very final level, and it is mostly written in French. In practice the universal c language together with sql describe was used to dig out the facts how to define a new sequence type. The well defined consraints mentioned earlier were very helpfull while doing the first sequence type definitions of our own. After doing the first definition of the kind, and learning what has to be defined, it is really a simple and flexible task to define new appeareances or redefine an existing one.

3. Data Flow

Let's take a look on how the data flows through the NEONS system.

If we start from the outermost layer on the load side we have the following: The data is loaded automatically by the load daemon processes from the directories specified by environmental variables. The load daemon processes are named moddap for the grid data and bdm_lanceur and bdm_load for the observations.

The environmental variable specifying the load directory for the observations is $BDM_LOAD. The files to be placed into this load directory can contain one or more observations each in a three-piece format; the header part describing the assosiative info (place, date, time, etc...) of the observation, the param portion containing the definition of the 25 float columns and finally the BUFR bitsream.
For the grid datatype, the environmental variable to specify the load directory is $BDAP_LOAD. The files must be in standard GRIB format, one or more messages per file.

On the data retrieve side, both C (and Fortran) function and unix command line level interfaces are currently available .
As input to these interfaces, the user has only to specify the criteria on which he/she wants the data to be searched; on which place or area one is interested, at which time or timerange, what parameters one is interested in, on what levels, etc... One doesn't have to consider details of the database or any other environmental configuration, its enough to specify the essential meteorological characteristics for the search.
As output, the unix commands getgribsfromneons and getbufrsfromneons produce a file in GRIB and BUFR formats (respectively) containing all the matching data.
The functional interface routines GetGribsFromNeons and GetBufrsFromNeons (C versions) will pass back to the user dynamically allocated arrays containing the matching data. The Fortran versions of these routines will require the output arrays being allocated by the calling routine. These functional layer routines will be further used to implement the next layer of interface, which will be object oriented and has meteorological features, such as time series handling, iplemented automatically within the classes provided.

What we have between these outer layers is basically two more layers. In the core of the system we have the core RDBMS system, providing the data store and a standard SQL interface for the next layer access routines.
The next NEONS layer consists of a set of proc functions interfacing to the core RDBMS with standard SQL. It will provide a set of primitive NEONS functions, such as opening/closing the database connection and reading/writing specific data. These routines will be used by the next outer layers to provide a more sophisticated interface as described in the previous paragraphs.

The valuable thing on the layered design of this system is, that one can easily switch the code on one given layer to something else. One only has to take care, that the new code will provide the same interface upwards and downwards as before.

4. To Be Continued . . .

A few remarks on the plans for the near future around this system.

On the data load side we are relatively happy with the system as it is. It is very flexible and has been working reliably for some months already.
However, we would like to have our SMS system being able to control the data load processes. This would integrate the NEONS data load processes into our existing operational resources. This means some development work for more batch oriented load processes that can be started from SMS to load predefined files and thus also controlled through SMS.

On the data retrieve side we are developing an object oriented interface on top of the 'C-functional' layer. This OO interface will be based on a MetBox object, that is a self descriptive and self sufficient box of meteorological data. MetBox object has relevant meteorological data features within it, e.g. missing data will be automatically filled into a requested time series (if desired, of course). This MetBox object is able to carry point, grid and other meteorological datatypes.
As this MetBox interface is independent of the hw/sw platform, network and other environmental aspects, the development of Client/Server applications based on MetBox is

very handy. We can bring the MetBox to the application almost 'on the back of a bird', and the application knows how to deal with the self-sufficient MetBox without any further connection to the provider of the data.

To store the grid data into NEONS we act in a bit different manner than they do in Meteo-France. We store the the GRIBs from the models in exactly the format, that the models have produced them, whereas in Toulouse the GRIBs loaded into NEONS are preprocessed into a unique form for Meteo-France. In our case, we have loads of applications already around, that want their GRIBs in the original form of the producer.
Actually what happens in our case is, that we always load the original GRIBs in the original format to the database, from where they will be automatically read by postprocessing processes to reformat them into out local units, geometries, etc... that suit better to our graphics interfaces and such. The postprocessing will add the new GRIBs in the new formats into the database for later access. And the original ones can be easily destroyed right after, if they are not needed by anyone.
This way of storing the GRIBs in their original forms raises the problem of how to find out how a certain parameter is represented by different models. Let's take the Large Scale Precipitation as an example:

	Hirlam/FMI	ECMWF	FMI Local
Param Identifier	62	142	62
Unit	KgM2	m	mm/10
Leveltype	GROUND	HEIGHT	HEIGHT

As you can see, the param ids, units and level types differ from producer to producer. So one would have to code loads of logic into the accessing functions to deal with this. And if for some reason the presentation of a given parameter would change at some point, we would have to modify all this coding.
We are developing a 'Universal Parameter Naming' convention to bypass this problem. The end user can request any parameter in exactly the form he find's most suitable for him/her, and the automatic parameter naming functionality will dig out the exact values with which the item has been stored into the database, including the conversion factors for units. This functionality is based on a couple of additional cross reference tables for the parameters and levels. Now that the dynamics of the parameter naming is in database tables, one doesn't have to modify codes, if a presentation of a parameter changes for any reason.

5. Summary

We have found it very easy to port the NEONS-Meteof system on our local environment.
NEONS-Meteof has been found to be dynamic, flexible and very suitable to fulfill our requirements for a transient met data store.
It has been running with HIRLAM and ECMWF gribs (+ postprocessing) and SYNOP, SHIP and one specific AWS observation for our sea areas in routine manner for a couple of months already.
It has proven out to be very reliable.
Based on the NEONS-Meteof, it has taken about half a man year to bring up a production ready transient data storage with a basic functional interface.