H13-75

CAMXRUNNER: A MODULAR ENVIRONMENT FOR EFFICIENT CAMX SIMULATIONS

Daniel C. Oderbolz[1], Şebnem Andreani-Aksoyoğlu[1], Iakovos Barmpadimos[1], André S.H. Prévôt[1], Urs Baltensperger[1]

[1]Paul Scherrer Institut, Villigen-PSI, Switzerland

Abstract: Air quality models are complex systems, which are challenging to run. In the future, complexity of air quality models will rise: more computing power offers the possibility of a higher amount of output, which in turn complicates data analysis. In the light of this, the traditional triad of pre-processing, simulation, post-processing must be rethought. An enhanced approach is to break up the sequential structure of this process, allowing for parallel processing. This can lead to a significant speedup of model runs and a more efficient use of computer resources.
Modular modelling systems, such as the Comprehensive Air quality Model with extensions (CAMx [*ENVIRON*, 2008]) offer various model setups for each model run. This creates a large number of possible permutations of settings and input data and stresses the need for automatic pre- and post-processing. If manual steps are involved in running a model system, reproducibility is hindered, on the one hand because of operator errors, on the other hand because of missing documentation of these manual steps.
CAMxRunner is a modular, extensible suite of bash scripts that automates whole model runs including pre- and post-processing and supports both sequential as well as parallel approaches. It was written with CAMx as a target platform, but it is general enough to support other models.
A run is controlled by a configuration file and eliminates manual steps. This approach is an important step towards efficient model runs.
The system keeps track of steps already executed, so runs can be suspended and resumed later.
Since the outcome of a model run critically depends on the programs used, CAMxRunner includes capabilities to manage different versions of all binaries needed. This includes a separation of different model versions and their support program as well as the separation of different compiler options (for example to support different platforms).
Especially for longer model runs, the use of CAMxRunner reduces the workload on the modeller but also assists the interpretation of the results by fully documenting each run in a configuration file.

Key words: *CAMx, Automation, Reproducibility, Accountability*

INTRODUCTION

Reproducibility is an important challenge for modellers across all scientific disciplines, mostly because it is easy to change the setup and the source code of a model, but hard to document these changes. We propose a new approach to modelling in general and air quality modelling with CAMx in particular and implemented these ideas in a complete end-to-end modelling system.
The reasons why it is often difficult to reproduce model runs, especially those by other groups are manifold: be it custom modifications to model code, special treatment of input data or simply undisclosed options. We believe that the current state approach to modelling harms the reputation of the whole community, because the ability to reproduce results of others is a central aspect of the scientific method.
CAMxRunner is a modelling system that initially grew out of the need to reduce as many manual steps as possible. Gradually, it became a full-fledged environment for modelling with CAMx. Even though it was designed to be used for CAMx, it is flexible enough to support other models – an implementation to support the meteorological model WRF (Weather Research and Forecasting) [*Michalakes et al.*, 2004] is planned in the near future.

Other authors, including [*Schwab et al.*, 2000] proposed similar solutions to the issue of reproducible model output. The approach by Schwab *et al.* is based on the Unix tool *Make* which is powerful but not user-friendly. Also, they did not consider the modelling system as a whole, so they do not address the management of the binaries and modifications thereof.

CAMxRunner is (in our opinion) a user-friendly tool that can manage the whole "lifecycle" of model data, from the generation of the input data to the production of plots.

We have used CAMxRunner for our day-to-day modelling needs as well as to allow beginners to use CAMx, which dramatically reduced the learning curve.

IMPLEMENTATION

CAMxRunner was written in Bash (>= 3.x) because CAMx is normally run using C-shell scripts, but C-shell does not offer support for functions. This should enable CAMx users to understand the code quickly and be able to change it to specific needs.
It was an important design goal that the system is modular and extensible, especially since CAMx is modular as well and pre- as well as postprocessing depends on the modules that are currently in use.

The system is built on a simple directory structure that is based on the distinction of modules for different models and their versions.

Figure 1: The directory structure of CAMxRunner

In Figure above, you can see that the subdirectory combination CAMx/4.42, 4.51 and 5.10 occurs in many different places to separate modules (these are bash-wrappers to call the binaries) and other components of the system.
To introduce a new model like WRF, the respective directories need to be created (the system can do this automatically) and be filled with the relevant modules by the user.
Templates for modules (in the templates/ directory) give users a head start for the further development of CAMxRunner.

A modelling system consists of many different binaries (pre- and postprocessors and the model itself), which poses some special problems if the same system should be used by different hardware platforms. At the Paul Scherrer Institut, we use the Andrew File System (AFS), a network filesystem to store our data centrally for all servers. Therefore CAMxRunner keeps binaries for different platforms separated, to avoid this issue. This is achieved using a logical naming convention that allows to infer the platform for which a binary was compiled.

In contrast to the "jobfiles" that ENVIRON supplies with CAMx, CAMxRunner strictly separates configuration data from code to run the model. In addition to this, it fully integrates pre- and postprocessing, so that at the end of a run, one is left with completely processed data, whatever postprocessing there is needed. That way, a CAMxRunner configuration file contains all settings relevant for the model run in question.
Often, configuration items (say, an input file path) depends on runtime data, such as the day currently simulated. To account for this, CAMxRunner offers so-called file-rules, which are arbitrary strings that may contain variables that are resolved at runtime. The system incorporates a multitude of variables (mainly date-related) that can be used in such rules. For example the rule *'$CXR_INPUT_DIR/tuv_$CXR_CAMX_CUSTOMER_winter$CXR_YEAR_S_week_CXR_WOY.out'* is resolved to /inputs/tuv_bafu_winter_06_week_1.out when simulating week 1 of winter 2006 for customer bafu.

Due to its modular architecture, it is straightforward to add new processing modules, be it for pre- or postprocessing. Since CAMxRunner normally runs non-interactively for obvious reasons, the only requirement for such a module is that it can be run without user intervention and does not for example require a graphical user interface.
Since model runs depend on many input files, there is the possibility of them changing without the modeller's knowledge, especially if a whole team provides input data. To account for this, CAMxRunner keeps a database of checksums (MD5 hashes [*Rivest*, 1992]) which allows to detect even single bit changes of files.

Another important design goal was to make full use of modern multi-core shared memory systems. The CAMx model itself already employs OpenMP [*Dagum et al.*, 1998] to exploit such systems optimally, but most pre- and postprocessors still use only a single core. To that end, CAMxRunner can resolve the dependencies of pre- and postprocessors which can then be executed in the proper order in parallel. This not only uses the computing system in a sustainable matter, but also reduces the total elapsed time required for a run.

To be able to infer the correct order to execute the tasks of a run, and to know which tasks are independent, reasoning on the dependency graph is needed.

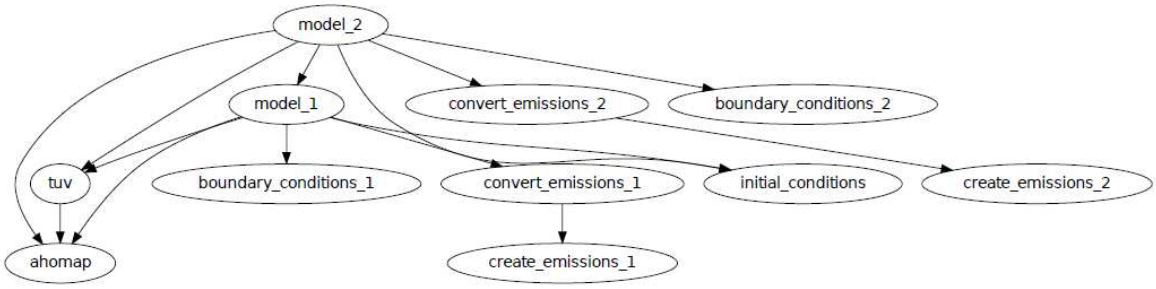

Figure 2: A sample dependency graph of a two-day run

Figure 2 shows an example of such a graph: nodes represent tasks ("*create_emissions_2*" is the task that creates emissions for the second modelling day) and arrows (edges) mean "depends on". In our example, the tasks *ahomap*, *initial_conditions*, *create_emissions_1 and _2*, *boundary_conditions_1 and _2* are independent of all other tasks and could be run in parallel first. Two approaches to determine if tasks are independent, include looking at the adjacency matrix of the graph and sorting the graphs nodes topologically [*Chartrand and Zhang*, 2005].

[*Hartel and van Harmelen*, 1984] have proposed the Unix command *tsort* to sort tasks topologically, allowing to resolve dependencies. CAMxRunner also uses *tsort*, since, as a by-product it also reveals any cycles within the graph (meaning that a task depends on itself, which is a serious error condition).

Currently, CAMxRunner includes support for CAMx 4.42, 4.51 and 5.10, which are properly separated by a general directory structure. At runtime, CAMxRunner determines which model and version to use and loads the appropriate modules for this setup. May modules are shared among versions, in such cases, UNIX softlinks (*ln –s*) are used to refer to the actual files, so that only one version needs to be maintained.

To keep track of any custom changes of the model, CAMxRunner includes a facility to store patches (files created with the UNIX tool *patch*) to the source code in a way that makes it easy for others to apply the same patch to their code as well. Generally, CAMxRunner offers a lot in the area of compilation – the system can for example ask the user predefined questions like "What is your maximum grid size in x-direction" in order to apply all relevant changes to the source code. The list of changes is available to the system at run-time, so that it can for example test in advance if a problem domain is too large to be processed by the current model binary.

At runtime, sophisticated check functions make sure that input file are present or that output files are absent (CAMxRunner will not overwrite existing files unless the user specifies this explicitly). CAMxRunner uses its logging facility to report information about the course of the simulation – the user can select which level of information (debug, verbose, information, warning, error) should be displayed on screen, written to a file or sent via email.
If a Mail2SMS gateway is available, this can be used to send SMS on error conditions – this is a feature we use here at PSI. Each step that was executed successfully is recorded, so that restarting a run that failed at some point is as easy as restarting a script (if fixing the problem is trivial).

From a programming methodology point of view, CAMxRunner uses a test-driven approach, as proposed by Beck [*Beck*, 2003]. This means that before new functionality is added, automated tests thereof are added to the system. This changes the focus of the programmer towards unusual situations that might occur and lowers the risk of errors. Currently, the system includes more than one hundred tests that can be called routinely. This also allows testing for any bad side-effect of new code quickly.

Finally, the code is formatted in a way that it is well readable and it is annotated in a way that NaturalDocs [*Valure*, 2010] can automatically produce HTML code documentation of the whole system.

CONCLUSIONS

We described an integrated approach to air quality modelling and presented CAMxRunner as a first implementation of these ideas. We believe that such a framework is beneficial for many model users and is not only limited to CAMx.
Also, we hope that the ideas outlined here as such find their way into many modelling teams, since most of the techniques have been around for a long time (the Unix tools used in this programs are all several decades old, but still very useful).

OUTLOOK

Starting with version 5, CAMx supports MPI (Message Passing Interface) as an additional parallel programming approach besides OpenMP. This allows CAMx users to take full advantage of large computer clusters that support only distributed memory. However, such systems normally use some kind of queuing systems (The European Centre for Medium-Range Weather Forecasts for example uses LoadLeveler™ by IBM) to accept batch jobs. Such systems use special files to describe a job. We are in the process of making CAMxRunner capable of producing such files automatically.
Also, we are working on the implementation of WRF in our system – with this, CAMxRunner would offer a complete Air quality modelling platform both for regulatory as well as for research use.

ACKNOWLEGEMENTS

This work would not have been possible had ENVIRON Inc. not publicly released their CTM CAMx. I thank all the employees of this company, especially Chris Emery for their support and valuable input. CAMxRunner includes code written by these people: PatrickLeBoutillier (we use his “test anything" implementation) and Chris Johnson (he provided some date functions).
This work has been partially funded by the Swiss FOEN (Swiss Federal Office for the Environment) under contract Nr. 06.9115.PZ/H094-1453.

REFERENCES

Beck, K., *Test-driven development by example*, Addison-Wesley Professional, 2003.
Chartrand, G., and P. Zhang, *Introduction to graph theory*, McGraw-Hill Higher Education, 2005.
Dagum, L., R. Menon, and S.G. Inc, OpenMP: an industry standard API for shared-memory programming, *IEEE Computational Science & Engineering*, *5*, 46-55, 1998.
ENVIRON, User's Guide, Comprehensive Air Quality Model with Extensions (CAMx). Version 4.50, ENVIRON International Corporation, Novato, 2008.
Hartel, P.H., and F. van Harmelen, Analysis of modular structures with respect to their interconnect topology, *Interfaces in Computing*, *2*, 81-92, 1984.
Michalakes, J., J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang, The weather research and forecast model: Software architecture and performance, in *11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, pp. 156–168, Citeseer, 2004.
Rivest, R., The MD5 Message-Digest Algorithm, in *Internet RFC (Request for Comments)*, 1992.
Schwab, M., M. Karrenbach, and J. Claerbout, Making scientific computations reproducible, *Computing in Science and Engineering*, 61-67, 2000.
Valure, G., The NaturalDocs Website, 2010.