22nd International Congress on Modelling and Simulation, Hobart, Tasmania, Australia, 3 to 8 December 2017
mssanz.org.au/modsim2017

Uchronia, a software module for efficient handling of multidimensional time series and use in ensemble forecasting

J.-M. Perraud [a], **R. Bridgart** [b], **J.C. Bennett** [b] **and D.E. Robertson** [b]

[a] *CSIRO Land and Water, Canberra, Australian Capital Territory*
[b] *CSIRO Land and Water, Clayton, Victoria*
Email: jean-michel.perraud@csiro.au

Abstract: Ensemble prediction techniques have been shown to produce more accurate predictions than single prediction runs as well as being able to formally quantify prediction uncertainty in a range of scientific applications. Statistically meaningful quantification of uncertainty can require very large ensembles, with associated increases in computation and data storage, particularly for predictions in the form of time series. In addition, the verification of statistical properties of an ensemble, such as the reliability of the ensemble spread, requires very long retrospective verification periods. This presents logistical and conceptual challenges for researchers and practitioners still transitioning from simulations based on deterministic, single instance realisations. A software system for handling such ensemble time series has to address many needs, notably: (i) retrospective ensemble predictions can quickly require several hundred gigabytes of data, and may need to be accessible from workstations or high performance parallel compute clusters; (ii) users should still have an interactive, responsive experience when processing data, with little concern for on-disk logistics - a well-known principle that remains unsatisfactory in many implementations; and (iii) data infrastructure must promote strong data identity and versioning to help sustain a reproducible simulation outcome.

This paper presents an ensemble time series software infrastructure that has stemmed from needs in streamflow forecasting research, with potentially much wider applicability. The core entities of time series and ensembles are implemented in portable C++ code, using template metaprogramming. This permits a unified but versatile implementation for handling time series of various dimensional complexity. Time series elements can thus range from an atomic value, typically numeric, all the way up to a time series of ensemble forecasts. Object-oriented design patterns are used to allow for RAM caching of large data. A C API permits convenient data manipulation from a variety of interactive higher-level technical computing languages such as Python and R. Time series can be accessed via a time series library. Time series creation and retrieval relies on string identifiers and metadata, rather than paths to data files which are prevalent. The libraries can bring otherwise disparate data into a consistent data set for the simulation or analytical purposes. We envisage the time series library facilities as a solid basis for use in conjunction with federated data provenance infrastructure.

Keywords: *Ensemble streamflow forecasting, time series, interoperability*

1. INTRODUCTION

Ensemble prediction techniques have been shown to produce more accurate predictions than single prediction runs as well as being able to formally quantify prediction uncertainty in a range of scientific applications. Statistically meaningful quantification of uncertainty can require very large ensembles, with associated increases in computation and data storage, particularly for predictions in the form of time series. In addition, the verification of statistical properties of an ensemble, such as the reliability of the ensemble spread, requires very long retrospective verification periods.

The semantic meaning of a large simulation and big data footprint is not an absolute but relative to the state of the art in a particular domain. Ensemble prediction techniques presents logistical and conceptual challenges for researchers and practitioners in hydrology still transitioning from simulations based on deterministic, single instance realizations.

In this paper, we present a software module for handling multidimensional time series data, named *uchronia*. This module stems from existing and nascent needs in ensemble prediction techniques, for use both in research and operational contexts. Usage in a research context, such as retrospective ensemble predictions, can quickly require several hundred gigabytes of space for data, and need to be suitable for a researcher's workstations but also scale to a high performance parallel compute cluster. Users should still have an interactive, responsive experience when processing that data, with little concern for on-disk logistics, something that remains unsatisfactory in many implementations. The data infrastructure must promote strong data identity and versioning to help sustain a defensible and reproducible simulation outcome.

Uchronia has stemmed primarily from work on a software for ensemble streamflow forecasting (Perraud *et al.* 2015). Since that publication it has been further engineered to fully decouple it from the modelling and simulation part of the software stack, and uchronia is now a standalone, modern, C++ library technically reusable in other domains dealing with multidimensional time series. Our intention is to have uchronia accessible by a broader community once licensing and governance arrangements are agreed open by project owners; in the meantime we invite the interested reader to contact the lead author of this paper.

2. OVERVIEW OF RELATED SOFTWARE

This section gives some background context that informed the design of uchronia, focusing on the existing software that was considered as candidate for reuse or design guidance in handling multidimensional time series for semi-distributed modelling and simulation. It is not a comprehensive review of software systems for all the facets of time series handling.

Popular data science programming environments have, by and large, adequate support for time series, especially point multivariate series. The R ecosystem has several time series handling packages (https://www.r-pkg.org/ctv/TimeSeries), notably *xts*, which can handle multivariate time series. In the Python ecosystem, the functionalities offered by the *pandas* (http://pandas.pydata.org) and *numpy* (http://www.numpy.org) packages have gained popularity and appear to be a *de facto* standard for general-purpose time series and/or multidimensional data handling. The multidimensional capabilities of numpy arrays is of particular interest to handle the multidimensional time series we are interested in. MATLAB has two main data types for time series handling (https://au.mathworks.com/help/matlab/time-series.html), *timeseries* and *tscollection*, with functionality that appear to be conceptually similar to those in xts. Both xts and numpy actually have core data handling written in C/C++, and were considered for reuse in early stages of work on uchronia. However, their features are either not generic enough or remain, to a significant extent, tied to their primary higher-level language, thus limiting interoperability and the uniformity of a user experience across several technical computing languages.

Traditionally, strong contenders for handling multidimensional data including time series for earth sciences are netCDF and HDF5. These primarily deal with file storage mechanisms. Abstractions are offered in some of the associated software packages such as the netCDF Java API (https://www.unidata.ucar.edu/netcdf-java), but remain to some extent bound to the storage layer. The lower level subsystems of Uchronia are in the majority using netCDF, and some architectural commonalities can be found elsewhere in uchronia, however direct bindings to netCDF have to remain confined to the lower software layers.

Curiously, in the realm of native C++ libraries, we could not find a main contender for general-purpose multidimensional time series handling. For instance, although the *Boost* software library (http://boost.org) has a large scope, it appears not to comprise of a fully-fledged subsystem dedicated to time series. The Boost MultiArray library is relevant to the representation of multidimensional time series, and might be considered in the future as one data storage option for uchronia.

3. SAMPLE USAGE

One of the primary use cases for uchronia is handling data input to, and resulting output data from, retrospective ensemble streamflow forecasts. This section illustrates usage focusing on direct data handling without a modelling engine but it is important to note that uchronia is also designed to offer first-class time series handling in modelling engines. One such case is the SWIFT2 modelling toolset in (Perraud *et al.* 2015). The present sample will use code snippets in R, but since uchronia is written in C++ with a C API, the same user experience is already available or possible from a variety of other interactive languages, such as Python and MATLAB.

Figure 1 is a small but realistic example showing how to load a data set of ensembles of streamflow forecasts. Uchronia promotes the use of data libraries and data identifiers, hiding the details of the storage mechanism. Currently time series libraries are described using a high-level description in YAML[1]. This is but one possibility; a pragmatic choice as a first step to start abstracting pre-existing file systems arrangement. Later in this paper we will mention further options.

```
library(uchronia)
my_lib_file <- '//server/path/to/ovens_library.yaml'
data_dir <- '//server/path/to/storage_root'
data_lib <- getEnsembleDataSet(my_lib_file, dataPath=data_dir)
getDataIdentifiers(data_lib)
# [1] "ovens_streamflow_forecasts_ens_13" "ovens_streamflow_forecasts_ens_15"
tsEnsTs <- getDataSet(data_lib, 'ovens_streamflow_forecasts_ens_15')
str(tsEnsTs)
# Formal class 'ExternalObjRef' [package "cinterop"] with 2 slots
#   ..@ obj :<externalptr>
#   ..@ type: chr "ENSEMBLE_FORECAST_TIME_SERIES_PTR"
print(geometryOf(tsEnsTs))
# $temporal
# $temporal$start
# [1] "2010-08-01 21:00:00 UTC"
# $temporal$time_step
# [1] "86400s (~1 days)"
# etc.
fcast_times <- timeIndex(tsEnsTs)
fcast_ind <- 200
fcast <- getItem(tsEnsTs, i=fcast_ind, convertToXts = TRUE)
plotXtsQuantiles(fcast[1:144,], title=paste("Streamflow forecast issue time ",
fcast_times[fcast_ind]), xlabel='Lead time', ylabel='m3/s')
```

Figure 1. Loading an ensemble of forecasts

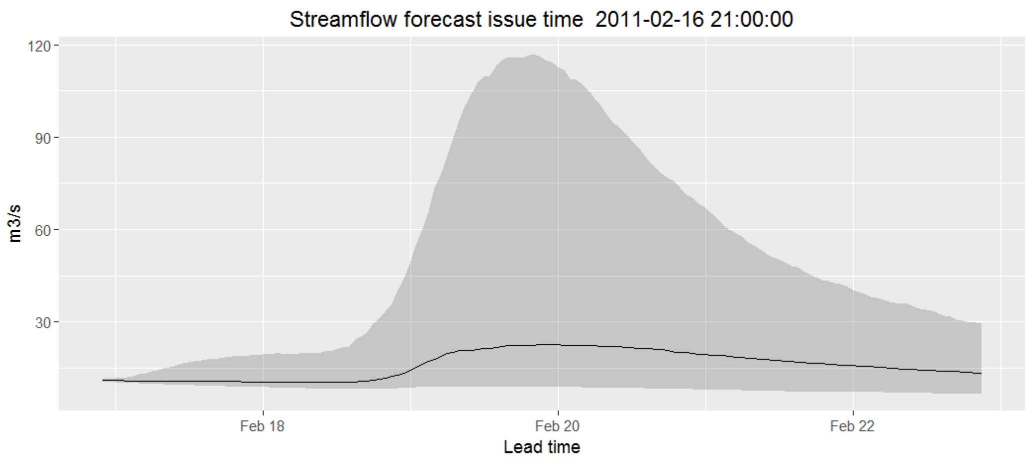

Figure 2. Visualising the ensemble of forecasts

[1] http://yaml.org/

Data from time series libraries are retrieved using string identifiers. Figure 1 illustrates how libraries can be queried to determine which identifiers are present in them. Hierarchical identification schemes are supported though not illustrated in this example. In this code sample we load a time series of retrospective ensemble forecasts, i.e. structurally a time series of ensembles of time series. The memory footprint of the data loaded is actually in the vicinity of two gigabytes, spread across files totaling ~30 gigabytes, stored offsite and accessed over a networked file system. All this is hidden from the user and the initial loading response time is immediate, for what is loaded is a lightweight proxy with lazy loading capability. This proxy, a C++ object, is referenced from the R language and memory is properly reclaimed upon disposal of variables in R.

Users can query the time series for its dimensionality, time spans, and time indices, and once again this is something that can be done without retrieving the bulk of the data. Data is fetched only when one of the items in the time series is needed for actual analysis or display.

4. SYSTEM DESIGN OVERVIEW

4.1. Architecture

Uchronia is primarily aiming to provide a robust, interoperable and efficient set of time series handling constructs for reuse in third party model simulation systems. It is written in modern C++ as this is a language that combines efficiency and modernity, still evolving, and there is a choice of high quality commercial and non-commercial integrated development environments. Figure 3 highlights the high-level components of uchronia but also, more importantly, emphasizes the boundaries of this library. Well known third party libraries such as netCDF and Boost are reused, and while some of their concepts are pervading (such as Boost POSIX date and time) they are usually not surfaced further up in the software stack. Uchronia does not, and will not, provide advanced data identity and provenance tracking in isolation; the design philosophy is to provide API entry points to use third party systems to provide plug-ins with such capabilities.

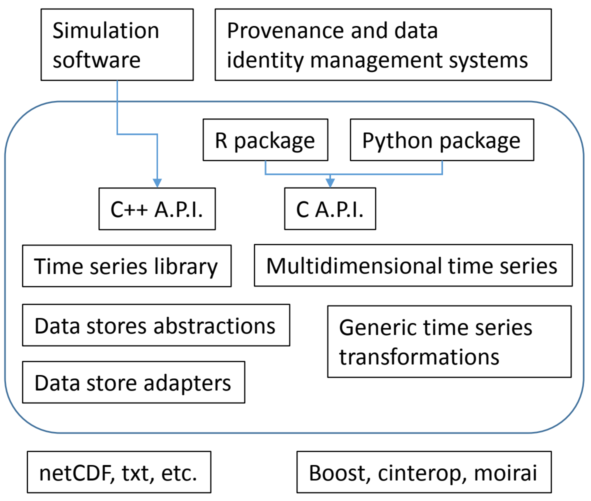

Figure 3. High-level architecture

The core features of a time series in uchronia borrows some patterns from TIME (Rahman *et al.* 2003), mostly in terms of time step handling and the "strategy" pattern for customizable data storage. Beyond these broad-brush similarities, there are differences and improvements, most notably with respect to multidimensional data.

4.2. Core time series structures

A salient feature of uchronia is its use of C++ template programming, albeit with some temperance. From its inception, the goal was to aim for a unique codebase for time series, ideally suitable for "time series of anything". The conceptual approach has been validated so far by use cases in ensemble streamflow forecasting, where the time series constructs went up to four dimensions with a "collection of time series of ensemble of time series", also known as "collections of ensemble retrospective forecast time series" in a more domain-specific streamflow forecasting parlance. Figure 4 shows the actual specialisation of template time series and collections (MultiTimeSeries) to these dimensions, with literally no additional code beyond the initial generic templates.

```
template <typename ItemType>
using PointerTypeTimeSeries = TTimeSeries < ItemType* >;
template <typename Tts = TimeSeries>
using MultiTimeSeriesPtr = MultiTimeSeries < Tts* >;
template <typename Tts = TimeSeries>
using ForecastTimeSeries = PointerTypeTimeSeries < Tts >;
template <typename Tts = TimeSeries>
using TimeSeriesEnsemble = MultiTimeSeriesPtr < Tts >;
template <typename Tts = TimeSeries>
using EnsembleForecastTimeSeries = PointerTypeTimeSeries < MultiTimeSeriesPtr<Tts> >;
```

Figure 4. Multidimensional template time series specialisations

The temporal axis of time series uses, by default, a concept of time step rather than a series of indexing date and times. Regular time steps support operators such as multiplication, division and equality testing to facilitate operations such as aggregation and disaggregation. Date and times are handled using Boost templates. Use cases to date have focused on UTC. Provisions for a more explicit handling of time zones is present where no ambiguity is arising, but it is worth noting that target applications have largely not necessitated this and we have deliberately pushed back developing this to a later date when clear use cases arise, as the topic is in our experience fraught with complications and issues.

While using C++ templates can be impressive to demonstrate as they stand, a naïve implementation quickly runs into memory limitation even on facilities generously endowed with RAM, with use cases already identified where the data footprint is in the terabytes territory. Time series can thus be constructed with a customisable storage policy, and missing value handling (Figure 5). A natural default for storage is in random access memory using standard C++ vectors. Storage policies with lazy loading or a windowing system have already been implemented for cases where the data is in one or several netCDF files. These cached data access patterns are commonplace in principles, though the implementation is far from trivial. It is worth noting that the design is not to have a general-purpose data caching system, but a customizable provision for targeting efficient caching mechanism tailored to the needs of the particular context and its data access patterns.

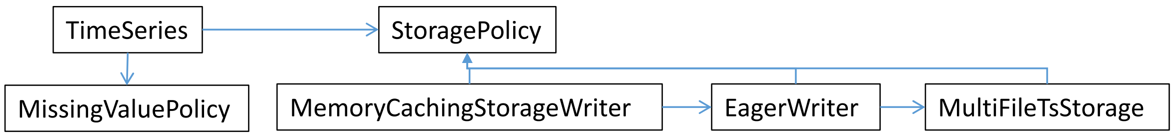

Figure 5. Composite storage and missing value policies for time series

4.3. Time series library and data providers

```
demo_streamflow_forecasts_ens:
  Type: ts_ensemble_ts
  Id: ovens_streamflow_forecasts_ens
  TimeStep: 24:00:00
  Start: 2010-08-01T21:00:00
  Length: 2153
  EnsembleSize: 1000
  EnsembleLength: 218
  EnsembleTimeStep: 01:00:00
  Storage:
    Type: multiple_nc_files_filename_date_pattern
    File: ./ovens/data_v2_{0}2100.nc
    Var: q_fcast_ens
    Identifier: 15
    Index: 0
```

Figure 6. custom data source provider

While progress has been made over the past decade to improve data handling and curation for computational research, effort remains to improve the separation of the business logic from the disk storage persistence, often compromised by mixing direct file system-based data file handling and simulation code. Uchronia features a concept of a time series library from which time series should be retrieved via string key identifiers. Time series libraries comprise a collection of objects describing time series data sources, customizable via object inheritance. One facet of these time series sources is shown in a YAML descriptor in Figure 6 which shows a three-dimensional retrospective ensemble forecast time series where data is spread across multiple files. Using YAML as a description is a compromise: it is structured for machines yet human-readable.

4.4. Application programming interfaces and interoperability

Uchronia can be accessed via two APIs: C++ or C (Figure 3). The C API may first appear as an unnecessary duplicate but has distinct advantages. C is a *de facto lingua franca* of native compiled code interoperability, whereas programming via the C++ API may tie you on to a particular C++ compiler. This is useful for surfacing in language bindings such as Python and R, and promotes a more consistent user experience. Embedding uchronia tightly as a central time series handling module for a simulation system is, on the other hand, better done via C++.

5. DISCUSSION

5.1. Retrospective lessons learnt

The inception of uchronia has been driven by the specific needs of research in ensemble forecasting, but also with the benefit of hindsight in using and implementing other time series handling software subsystems such as TIME (Rahman *et al.* 2003). Some classes in uchronia thus bear some similarities with TIME, notably in terms of time step handling and the “strategy” pattern for customizable data storage, wherever the existing approach was suitable. However most classes in uchronia were written from the ground up using template programming or metaprogramming, something that, initially, was technically not feasible for TIME. Template programming is one key technique permitting versatility for uchronia, where judicious. We mostly limit type parameters to one in classes, often the element type in a series or ensemble. While the C++ literature offered examples where policy/strategy patterns (in our case missing values and backend storage) were implemented template type parameters rather than polymorphic arguments, this proved overcomplicated or even counter-productive to do so in uchronia.

5.2. Opportunities for data systems integration

The architectural description in Figure 3 made clear that uchronia is covering a specific part of data provision and needs to be coupled to other data management systems to provide solutions such as the system described in (Yu *et al.* 2015b). netCDF is a natural candidate for persistent storage of multidimensional data, and already the majority of the storage layer under uchronia. netCDF files read/written by uchronia already follow conventions devised for ensemble forecasts, with a set of metadata attributes. These could be complemented with Linked Data semantic information such as devised in (Yu et. al. 2015a), or a new set of features in uchronia located higher in the software stack, and thus not limited to netCDF.

Data identity is key to the sustainability of a provenance and uchronia, by design, already promotes the use of identifiers of varying granularity for data items. The mechanisms already in place in uchronia time series libraries are a pragmatic first step away from *ad hoc*, historical reliance on file systems paths as data identifiers. We intend in the future to explore and apply designs such as that in (Golodoniuc et al 2017) to couple uchronia data provision with robust and persistent data identification mechanisms. A related topic of interest is enhancing the provenance tracking capabilities (see Car *et al.* 2014 for instance). The application of such provenance methodology has a scope that intersects with but is beyond that of uchronia.

6. CONCLUSION

Uchronia is a library written in modern C++ that arose from the needs of an ensemble streamflow forecasting software, and is technically reusable in other domains dealing with multidimensional time series. Uchronia can play a role at the interface between infrastructure storing and serving data and time stepping simulation software, and we identify opportunities for data systems integration to foster reproducible research. Our intention is to have uchronia accessible by a broader community once licensing and governance arrangements are agreed open by project owners; in the meantime we invite the interested reader to contact the lead author of this paper.

ACKNOWLEDGMENTS

This work is carried out in the CSIRO Water for Healthy Country National Research Flagship and is supported by the Water Information Research and Development Alliance between CSIRO and the Australian Bureau of Meteorology.

REFERENCES

Bennett, J. C., Robertson, D. E., Shrestha, D. L., Wang, Q. J., Enever, D., Hapuarachchi, P., and Tuteja, N. K. (2014). A System for Continuous Hydrological Ensemble Forecasting (SCHEF) to lead times of 9 days, *Journal of Hydrology*, 519, 2832-2846. DOI:10.1016/j.jhydrol.2014.08.010

Car, N.J.; Stenson, M.P. and Hartcher, M. (2014). A Provenance Methodology And Architecture For Scientific Projects Containing Automated And Manual Processes, *11th International conference on hydroinformatics*, http://academicworks.cuny.edu/cc_conf_hic/57

Golodoniuc, P., Car, N.J. and Klump, J. (2017). Distributed persistent identifiers system design, Data Science Journal, (vol. 16), Ubiquity Press. DOI:10.5334/dsj-2017-034

Perraud, J.-M., Bridgart, R.J., Bennett, J.C. and Robertson, D.E. (2015). SWIFT2: High performance software for short-medium term ensemble streamflow forecasting research and operations, *21st International Congress on Modelling and Simulation*, Gold Coast, Australia, p. 2458-2464. http://mssanz.org.au/modsim2015/L15/perraud.pdf

Rahman, J.M., Seaton, S.P., Perraud, J.-M., Hotham, H., Verrelli, D.I., Coleman, J.R. (2003). It's TIME for a new environmental modelling framework. In: Post, D. A., (Ed.), *MODSIM 2003 International Congress on Modelling and Simulation*, p. 1727-1732. http://www.mssanz.org.au/MODSIM03/Volume_04/C05/03_Rahman.pdf

Yu, J., Car, N.J., Leadbetter, A., Simons, B.A., and Cox, S.J.D. (2015a). Towards Linked Data Conventions for Delivery of Environmental Data Using netCDF, *ISESS 2015: Environmental Software Systems. Infrastructures, Services and Applications* pp 102-112. DOI:10.1007/978-3-319-15994-2_9

Yu, J., Hodge, J., Leighton, B, Seaton S., Vleeshouwer, J., Tickell, S. and Car, N.J. (2015b). Flexible and modular visualisation and data discovery tools for environmental information, *21st International Congress on Modelling and Simulation*, Gold Coast, Australia, p. 1317-1323 http://mssanz.org.au/modsim2015/F8/yu.pdf