Accessing External Data on Local Media and Remote Servers Using a Highly Optimized File Reader Library

Dipl.-Ing. Jörg Rädler   Dipl.-Ing. Manuel Ljubijankic   Prof. Dr.-Ing. Christoph Nytsch-Geusen   M.Sc.Dipl.-Ing.(Fh) Jörg Huber

Berlin University of the Arts / Universität der Künste Berlin (UdK)
Hardenbergstrasse 33, 10623 Berlin, Germany

jraedler@udk-berlin.de   manuel@udk-berlin.de   nytsch@udk-berlin.de   jhuber@udk-berlin.de

Abstract

ncDataReader2 [1] is an open-source solution for the efficient interpolating access to external data sets. The library of C-functions can be used with different applications and works well with Modelica. Data sets can be easily accessed as continuous functions using different interpolation and extrapolation methods. The application range covers reading generated or measured data, the integration of simulation results from Modelica or other systems and the validation, parametrization and optimization of models using external data. Data sources may be local files or remote servers. Using the netCDF file format [2], the DAP network protocol [3] and different optimization approaches the data access can be surprisingly fast, even for large remote files with many variables containing millions of values.

1 Introduction

Getting external data into a simulation model is an important task for a lot of applications: buildings and energy plants are exposed to weather factors, complex models need to be validated with measured values and some simulations require results from other simulation runs.

The access conditions can vary significantly. A dense grid of data can be interpolated in small or in large intervals, and so can a wide grid. A large dataset may be evaluated only in one point to compute initial values or interpolated a million times, moving backward, forward or randomly on the x-axis. For some of these conditions and small amounts of data the *Table*-like classes of the Modelica Standard Library are a good choice, but for different application scenarios the ncDataReader2 offers some real advantages:

- very fast, even with large amounts of data
- load on demand (only needed data is read and processed)
- low memory consumption (adjustable, suitable for embedded simulations)
- clever caching mechanisms, tunable for different access characteristics
- different interpolation and extrapolation methods
- offset and scaling of values for unit conversion and memory-efficient storage
- API[1] (ANSI C) and data files work the same way in Modelica systems and other applications
- data can be accessed locally or with a highly efficient network protocol (DAP)

Although used mainly for 1D data sets the library includes basic support for variables depending on two dimensions (scattered 3D-points)[2]. This paper will focus on the 1D functions.

1.1 History and Development

The development of the file reader library started more than 10 years ago as a tool for the DAE simulation system SMILE. Until now it was constantly improved and tested with SMILE [5], ANSYS CFX [6], the Modelica systems OpenModelica and Dymola and with proprietary applications.

ncDataReader2 is open-source software, everybody is invited to use and improve it under the terms of the GNU LGPL [7].

1 API - Application Programmers Interface: the data and functions available for the programmer
2 using the csa library from [4]

1.2 netCDF – the file format

netCDF is a binary file format[3] and a program library developed for large amounts of multi-dimensional geoscientific data. The big advantage over other formats is the ability to access pieces of data without reading whole data sets or even whole files. netCDF files are self-describing and may contain structured data of different dimensions. This makes a very good format to archive numerical data and a perfect base for a file reader used in DAE simulations.

1.3 Interpolation and Extrapolation

ncDataReader2 includes the interpolation methods Akima (most used), linear, discrete and smoothed steps[4] (see figure 1). Akima interpolation is a cubic method that gives smooth results (C1-continuity) without the tendency to overshoot. In contrast to classical cubic spline interpolation the points have only local influence, which perfectly complements the local access in netCDF. To get an interpolated value only some of the neighbouring points have to be read and processed after the search for the matching interval.

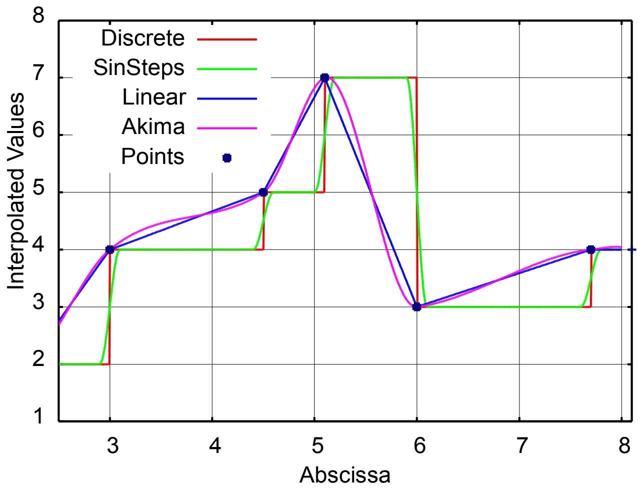

Fig 1: Interpolation methods

Extrapolation methods are implemented as periodic or natural (depending on the interpolation method).

3 Recent versions of the netCDF format are based on the HDF5 file format which is now used in MATLAB and many other applications.

4 Adjustable continuous approximation of discrete characteristics with C1-continuity, using linear parts and *sin*()-functions. Strictly speaking this is no real interpolation since the points are often not hit.

1.4 Tuning and Optimization

Variables may be fully loaded at initialization time, loaded in chunks of a specific size or as single values on demand. Three different caches may be enabled and changed in size:

- a lookup cache stores results of the interval search,
- a parameter cache holds the parameters of the linear or cubic function of an interval (both for successive requests of nearby values) and
- a value cache contains the last computed values (for successive requests of the same values).

The effect of these optimizations strongly depends on the access characteristics but may give a speedup factor of 100 and more in some cases.

The methods for interpolation and extrapolation as well as all parameters regarding loading, scaling and caches are preset to reasonable default values. All settings may be adjusted using attributes in the netCDF file or with the C API (full API only, see below).

Performance Example

The effect of clever using the optimization methods can be demonstrated with the example BigFile contained in the library. A data file with a size of 840 MB contains 10 variables each with 10 million random values. A Modelica class integrates the interpolated values of two of these variables (Akima method) over a sub-range of the abscissa. The result of the 500000 time steps is written to the result file.

With OpenModelica using default settings on a common PC[5] the program finishes in about 5 seconds! This includes the complete run with initialization, data reading, more than one million interpolations, numerical integration and writing of the results. Some other approaches would need much longer just to load the data file.

This performance is achieved by a combination of the lookup and parameter caches and by loading the data in chunks on demand.

5 OpenModelica 1.8.1, Ubuntu Linux, Intel Core2 Duo E7200@2.53GHz, 4GB RAM

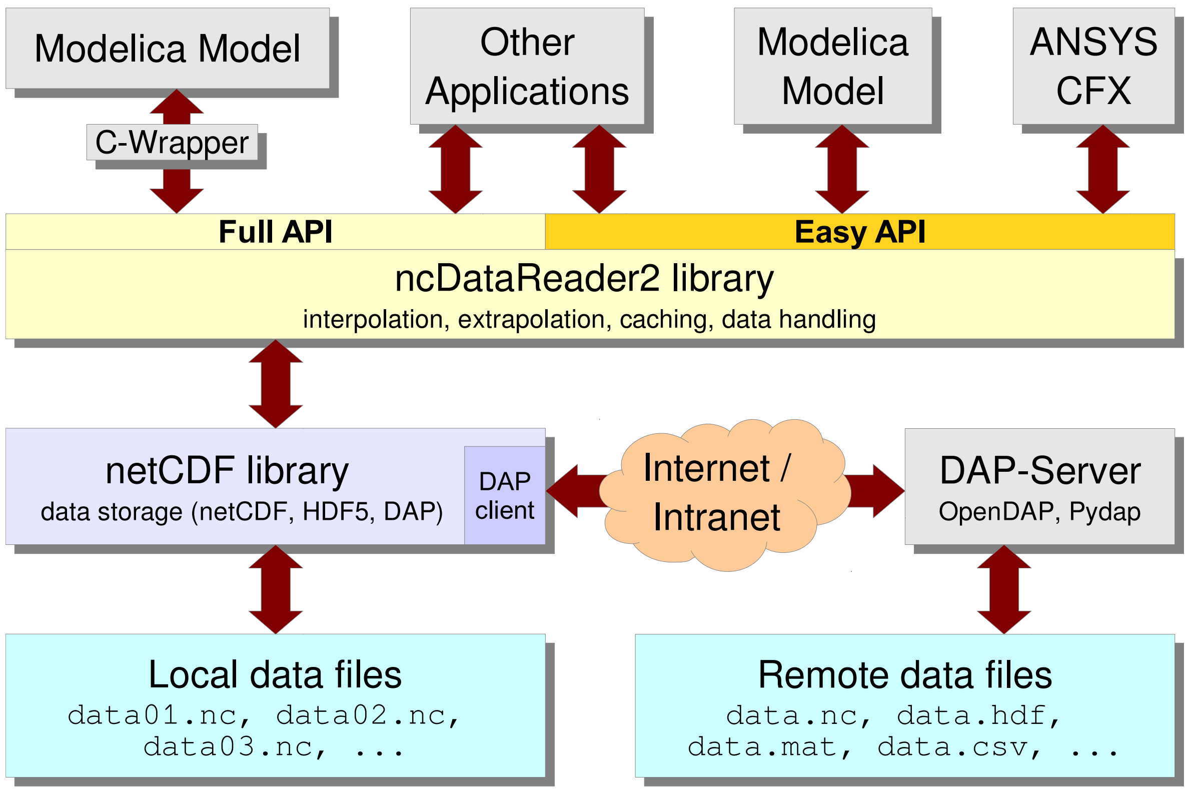

Fig 2: Different ways of using ncDataReader2 with Modelica and other applications

2 Modelica API

ncDataReader2 offers a full API and a so called easy API. The latter limits the configuration options and requires compliance to some restrictions, but it can be used in Modelica without writing C code. The full API is slightly faster and offers access to all options, but uses data types not available in Modelica. Therefore it requires adapted external functions and a bit more programming.

Both methods require the prior installation of the libraries ncDataReader2 and netCDF (which may depend on other libraries). The names of the files actually needed depend on the operating system, simulation software and compiler[6].

Most Linux distributions already contain the required packages for netCDF. For Windows precompiled files (.dll, .lib, .h) are provided by the developers.

The structure of the different APIs and libraries is shown in figure 2.

6 The search for files by the compiler on Windows systems may be confusing. Copy all files to the current working directory if in doubt.

2.1 Easy API

The Modelica package *NcDataReader2* contains definitions of all functions of the easy API. A very basic example demonstrates the usage:

```
model NcEasyTest
 import nc = NcDataReader2.Functions;
 parameter String fn = "etest.nc";
 Real t;
 Real a = nc.ncEasyGetAttributeDouble(
    fn, "", "start_value");
 equation
 t = nc.ncEasyGet1D(fn, "v1", time);
 der(a) = t;
end NcEasyTest;
```

Two functions of this package are used here:

- `ncEasyGetAttributeDouble` reads a scalar attribute to initialize `a`. The first and third arguments are the names of the file and the attribute. The second argument may be a variable name or empty (to use a global file attribute). Similar functions exist for attributes of integer and string types.
- `ncEasyGet1D` returns the interpolated value of the variable `v1` at the point `time`. A similar function for 3D-points exists.

At the first call to `ncEasyGet1D` the file is opened, the abscissa and dimensions of the variable are determined, optional attributes are evaluated and internal data structures are created and stored for later use. Subsequent calls with the same file name and variable name reuse these structures. Different variables depending on different abscissas in different files can be used at the same time.

2.2 Full API

The full API can only be used in C, not in Modelica. To utilize this API wrapper functions are required to hide the complexity from Modelica. The function definitions are split up in two parts:

a) A C-file which defines wrapper functions with simple interfaces (arguments and return values) to be used in Modelica. Inside these functions the full API may be used. A common case is to have one function with initialization code and one small function for each variable to return the interpolated values. This can usually be done within a couple of lines. All settings and options of the library may be changed in this file.

b) A Modelica file containing mappings of the C-functions to Modelica functions (usually 1:1). This includes the number and types of arguments and return values.

Although these steps are quite simple, an example would be too big to show here. Please have a look at the example in *NcDataReader2.Examples.FullAPI*.

3 Preparation of Files

Converting data into the netCDF format may be the hardest task for users without prior knowledge of netCDF. There exists no general graphical tool for this job, but besides command line tools for the conversion of an ASCII-based format there are libraries for all major programming languages (C, C++, Java, Python, Perl, Octave, MATLAB, …) and platforms.

A new project [8] from Microsoft Research provides a .NET-API, a graphical data viewer, command line tools and a plug-in for MS EXCEL to read and manipulate netCDF files on Windows systems.

The favourite method of the authors is scripting in Python. A lot of file formats can be read with Python, and consistency checks and unit conversions may be included in a script. The conversion of a simple CSV file can be done within 7 lines of Python code. This method works on all platforms.

When using a DAP server the conversion may be omitted entirely for some file formats.

4 Data Server with DAP

DAP is a protocol for the transport of multidimensional gridded data over networks. It is based on HTTP, but allows the request and the transport of specific parts of a file in different formats. DAP servers are able to serve requests like "*send me the values 1500...2000 of the variable 'temperature' in the file 'data.mat' converted to CSV format*". Clients can browse and request data with ease and efficiency. Data formats are converted on demand by the software (supported formats depend on the actual implementation).

Since newer versions of the netCDF library implement the client side of the DAP protocol, a DAP server can be used with ncDataReader2 like a local file, just by replacing the file name with an URI.

For Modelica users this combination offers a lot of options. External data used by simulations can be hosted on different servers worldwide. During simulation, only the actually needed parts are transferred to the simulation system. This ensures the up-to-dateness and the consistency of data across simulations and allows the cooperation of different institutions without sending complete files.

4.1 DAP server at the UdK

A new server at the Berlin University of the Arts (UdK) was installed for this purpose. It provides different kinds of data to a research group:

- Weather files for different locations worldwide, generated with METEONORM [9] and converted to netCDF (see 5.1).
- Data from the monitoring of a photovoltaic system located at the UdK main building. The data is read from the monitoring hardware and stored in netCDF files in regular intervals (see 5.3).

- Results from different simulations of the research group.

The server runs on common PC hardware using Linux, Apache and the Pydap [10] software.

5 Current Applications in Research

5.1 Reading Weather Files

Data sets with weather parameters were the first application for the data reader and still are most used. Thermal building simulation and simulations of solar systems require reliable information about the environmental conditions as functions of time. These conditions include:

- temperature, pressure and moisture of the air,
- wind speed and direction,
- direct and diffuse radiation, cloud cover.

For the evaluation of the radiation on different surfaces the position of the sun is needed, which can be calculated from the latitude, longitude and time zone of the location.

All this information can be easily stored in a netCDF file. Over the years some conventions regarding the file structure, the units and the names of variables and attributes have evolved and proved to be useful.

All this data is read and processed by a Modelica class (*DataWeatherNetCDF*). With the file name or URI as a parameter of this class the complete environmental conditions of a location may be set and changed. The contained quantities and some derived quantities are available as continuous functions of time. Common problems like negative radiation values caused by the cubic interpolation are handled. For different oriented surfaces the radiation values will be converted by a helper class (*RadiationTransformer*). All these classes are now part of the new Modelica library BuildingSystems [11] which is developed by the authors.

Generating Files

The files may be created from different data sources. The authors mainly used weather information from the TRY/Testreferenzjahr [12] and data sets generated with the application METEONORM. The latter let the user define own ASCII-based export formats, which can be easily converted to netCDF by a Python script. Our script now includes consistency checks, unit conversions, preparation for periodic extrapolation and much more.

With this method a library of weather files for different locations is built and expanded. The files reside on the DAP server (see 4.1) and are accessible by the whole research group. Data for new locations or new conditions can be generated and made available within minutes.

The time grid of most data files is equally spaced in hourly steps covering one year, but the software stack (DAP, netCDF, ncDataReader2, Modelica classes) works perfectly with different and variable steps and in other scales.

5.2 Loose Coupling of ANSYS CFX with Dymola

A research project at the UdK covers the co-simulation of a solar thermal plant. For pre-studies of a use case the ncDataReader2 is used for loose coupling. It reads the results of a Modelica simulation into the boundary conditions of a CFD[7] simulation with ANSYS CFX.

The main components of the plant are:

- an evacuated tube collector,
- a hot water storage and
- an external plate heat exchanger, transferring the produced thermal energy from the solar loop to the storage loop.

By using a two-point-controller the solar pump and the storage pump are simultaneously switched on. The system is modelled with Modelica, most components are based on the Modelica library *BuildingSystems* for thermo-hydraulic network simulation. The weather data is provided by the technology described in the previous section.

The storage model (marked in Fig. 3) can be either implemented in Modelica (1D) or in CFX (3D). The co-simulation of Modelica and CFX is described in [13]. It gives more accurate results regarding the details of the storage, but it runs much slower than the pure Modelica simulation.

Additionally stand-alone CFX simulations of the storage component were needed in the project. One

7 Computational Fluid Dynamics

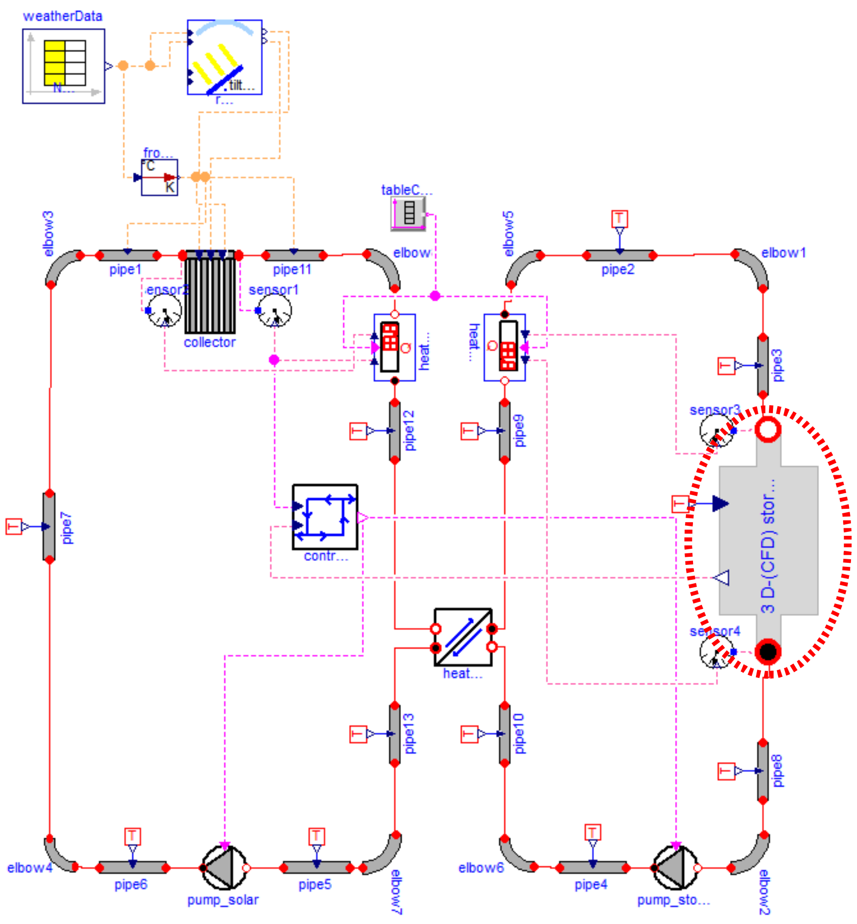

Fig 3: Modelica view of the co-simulation model of the solar thermal plant

of the questions to answer was the best resolution of the grid for the 3D model under typical conditions in the solar system. A high resolution will slow down the simulations, a wide grid will not reach the desired accuracy. A complete co-simulation model proved to be too slow to study this point in detail.

For a transient stand-alone CFX simulation of the thermal storage some boundary conditions are necessary to describe the installation situation. It would be possible to generate the time-dependant conditions with C-functions emulating the behaviour of the whole system, but the effort for this would be enormous. At this point it's much more comfortable to use the results from the system simulation with the simple storage model implemented in pure Modelica.

Dymola creates a result files in MATLAB format during the simulation. The structure of this file is quite complex, but can be read and converted with different tools. One is the Python package DyMat [14] which directly exports variables to different formats including netCDF.

The CFD model needs values for three quantities:

- inlet mass flow,
- inlet temperature,
- outlet pressure.

The time series for these variables can now be saved in a netCDF file.

ANSYS-CFX provides an API to implement dynamic conditions as Fortran functions. Since it is possible (but tricky) to call C-functions from this Fortran-API, ncDataReader2 can be used from this API with a small wrapper file to provide the values as interpolated functions of time.

The complete workflow is now:

a) Run simulations in Modelica using the pure Modelica storage model.

b) Convert the required results from the mat-file to netCDF format using DyMat.

c) Run CFX simulation of the complex storage model, reading boundary conditions from the netCDF file using ncDataReader2 and a Fortran wrapper.

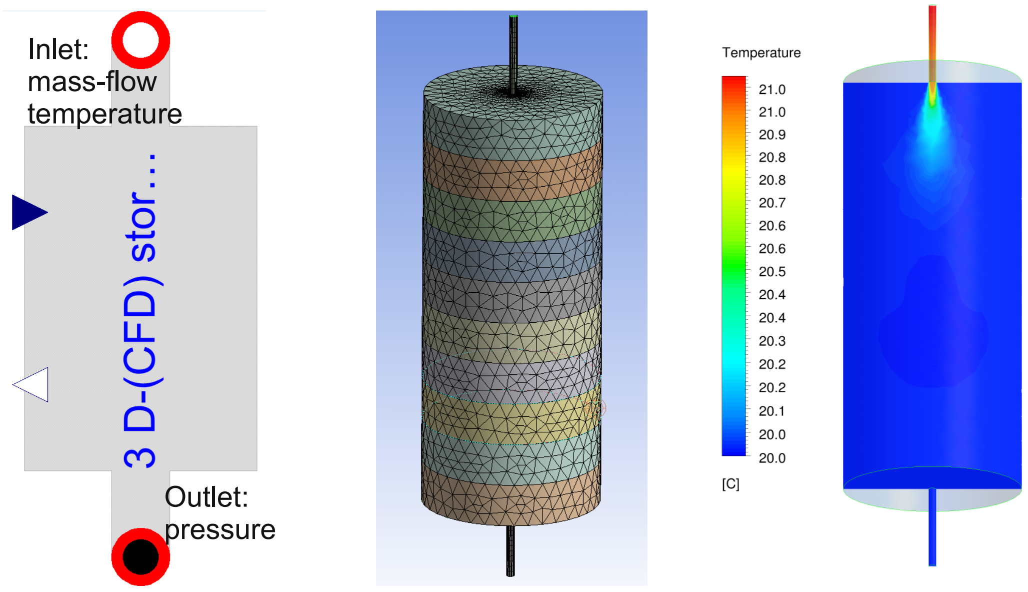

Fig 4: Storage model: a) Modelica connection component for the 3D model, b) grid of the CFX model, c) example of a temperature field using boundary conditions from the Modelica simulation

With this technology a stand-alone ANSYS-CFX simulation for the thermal storage can be started with dynamic adapted boundary conditions after each time step. Thus the CFX model is embedded in the same environment as the Modelica storage model in the system simulation for the solar thermal plant before.

This made it possible to research and tune the CFX model with respect to grid resolution and other parameters under typical conditions. Similar conditions appear in the real co-simulation which is the main topic of the research project [15].

5.3 Parametrization of the Model of a Photovoltaic Plant

The ncDataReader2 was used for the integration of measured data from a real photovoltaic (PV) system into a simulation model of the system. The complete field has an electrical power output (peak) of 15.5 kW and is located on the roof of the UdK Berlin main building. The measured values such as air and module temperature, solar irradiation, electrical output are used as climate boundary conditions of the Modelica system model and as comparison values for the model validation (see Fig. 5 and 6).

The Modelica model was configured by the use of the *BuildingSystems* library. One of the three strings of the photovoltaic field was modelled by the assumption of 22 serial connected PV modules. Each module (Type TSM-PC05) has a peak performance of about 230 W, so the total peak performance of the

Fig 5: Photovoltaic system on the roof of the UdK main building with sensors for solar irradiation, wind speed, temperatures of air and modules

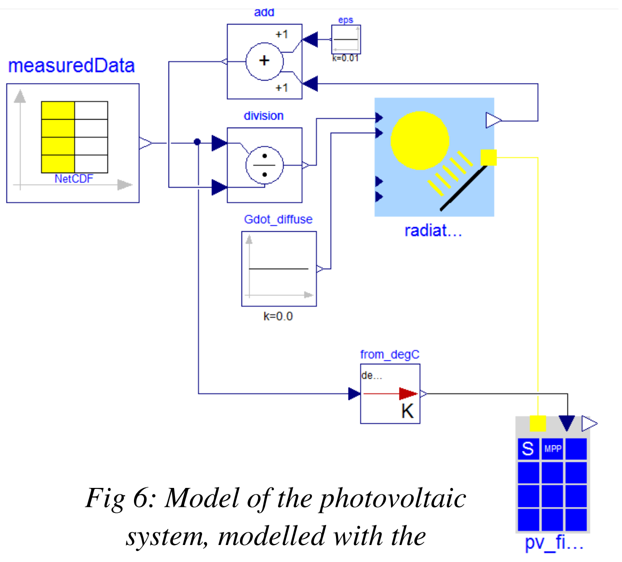

Fig 6: Model of the photovoltaic system, modelled with the BuildingSystems library

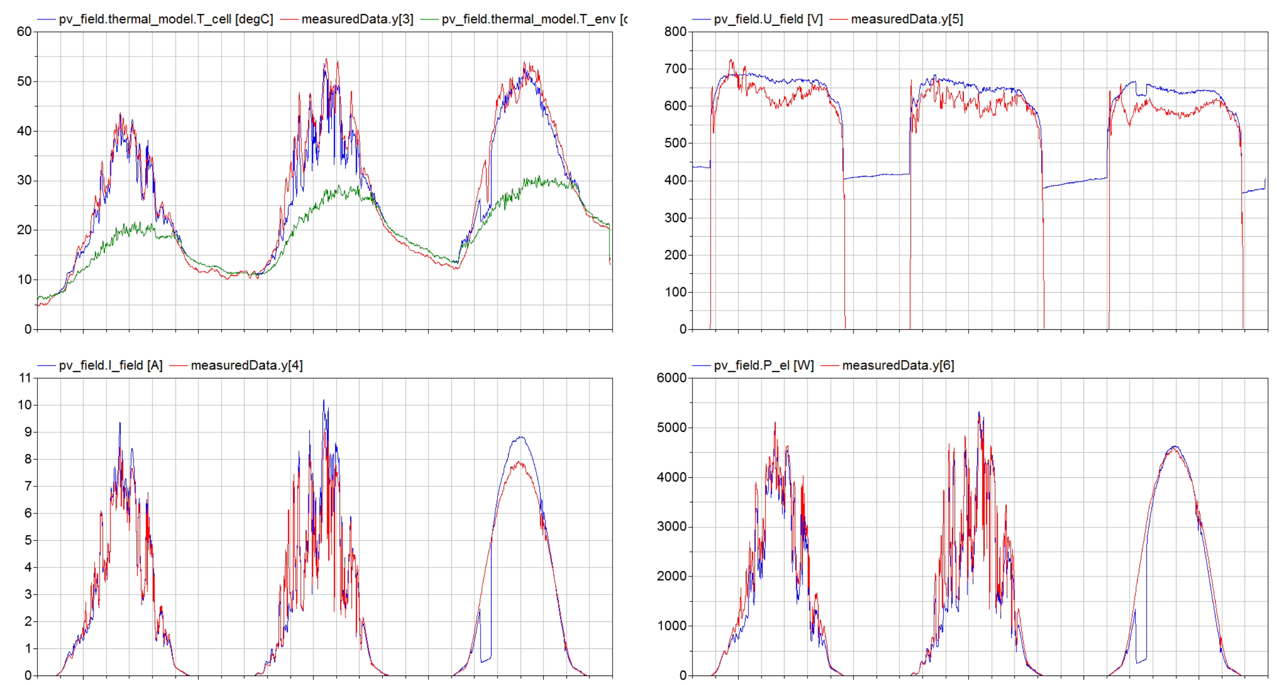

Fig 7: Comparison of measured and simulated values for three summer days:

a) cell temperature: simulated (blue), measured (red); measured air temperature (green)
b) string voltage: simulated (blue), measured (red)
c) string current: simulated (blue), measured (red)
d) string power: simulated (blue), measured (red)

string is about 5.060 W. The simplified model of a PV module of the *BuildingSystems* library was used, which works with an electrical one-diode model and an empirical thermal approach for the calculation of the cell temperature, depending on the air temperature of the cell environment [16].

Figure 7 shows the measured and the simulated values (temperatures, voltage, current and electric power) of the string of 22 PV modules during three summer days. All quantities have similar values for the real PV plant and for its simulation model. The cell temperature runs up two 20°C higher than the environment air temperature. Also the string voltage values are similar during the sunshine, in which the simulated values are higher than the measured values, the same goes for the current. After sunset the values of the simulation model are greater than zero, which is only a result of the modelling approach.

During the first two days the generated electrical power reaches the peak power for a short period. The collapse of the calculated electrical power during the morning hours is caused by a shading of the radiation sensor. The simulated performance drops because of the measured radiation, but the real measured performance is not affected. The position of the sensor will be moved to a better place in the future.

It is typical for a one-diode model that the voltage and current values are higher than the real values, because a part of the internal losses of the PV cells is neglected. Therefore a constant correction factor is used in this model for the calculation of the power from the voltage and current. This factor is a model parameter that depends on the real qualities of a module (materials and construction). Unfortunately it can not be derived easily from the properties that are usually known.

With simulations using measured values of the real system this factor can be approximated. For the three days of the shown configuration a value of 0.82 proved to be ideal to bring the calculated electrical power close to the real (measured) values.

The plant is monitored permanently and all values are archived on a data server (see 4.1). This makes it possible to adjust the correction factor of the model with simulations using recent measurements. This task can even be done fully automated.

6 Conclusions and Outlook

The library was used for more than 10 years in a wide range of applications. It can be used with Modelica and other systems to access data in different ways. In combination with a DAP server it offers the remote access to different data sources.

The most used application today is reading weather data in Modelica simulations of buildings and solar systems. But it is easy to use the library for other purposes and with different software packages.

Although the library is in a stable state there are some possible improvements for the future:

- better integration with Modelica runtime systems (e.g. error handling and reporting),
- supplying information on derivatives of functions for improved integration performance,
- implementing optimizations for special cases like equally spaced grids,
- providing better tools for the conversion and preparation of data files,
- possibly including the library and its dependencies in Modelica systems (Dymola, OpenModelica, jModelica) to avoid the complex installation process on the different platforms by the user,
- finding a better name for the project. :-)

References

[1] ncDataReader2: http://j-raedler.de/projects/ncDataReader2

[2] netCDF: http://www.unidata.ucar.edu/software/netcdf/

[3] DAP/OpenDAP: http://www.opendap.org/

[4] CSA: http://code.google.com/p/csa-c/

[5] Ernst, T., Klein-Robbenhaar, C., Nordwig, A., Schrag, T.: Modeling and simulation of hybrid systems with SMILE, in: Informatik, Forschung undEntwicklung, 15:33-55, 2000

[6] ANSYS CFD: http://alturl.com/m8fpb

[7] LGPL: http://www.gnu.de/documents/lgpl-2.1.en.html

[8] Dmitrov: http://alturl.com/g8wzk

[9] METEONORM: http://www.meteonorm.com

[10] Pydap: http://pydap.org

[11] BuildingSystems: http://www.modelica-buildingsystems.de

[12] TRY: http://www.dwd.de/TRY

[13] Ljubijankić, M., Nytsch-Geusen, C., Rädler, J., Löffler, M.: Numerical coupling of Modelica and CFD for building energy supply systems, in: Proceedings of the 8th International Modelica Conference, 2011

[14] DyMat: http://j-raedler.de/projects/DyMat

[15] Ljubijankić, M., Nytsch-Geusen, C., Jänicke, A., Schmidt, M.: Advanced analysis of coupled 1D / 3D simulation models by the use of a solar thermal system, in: Proceedings of the Building Simulation, 2011

[16] Nytsch, C., Quaschning, V., Scholtz, G.: Photovoltaik Modelle für die Simulationsumgebung SMILE, in: Tagungsband: 15. Symposium Photovoltaische Solarenergie in Staffelstein, OTTI-Technologiekolleg, Regensburg, 2000