DOE GMLC Project 1.4.18 “Computational Science for Grid Management”

Computational Framework Design for Power Grid Analysis

November 2016

Introduction

The computational framework aims to provide a seamless workflow that links data to computation to visualization so that a problem can be easily expressed in a way compatible with the solvers. This framework would also enable software compatibility such that application development can be more efficient. The two major factors driving the need for such a framework are the increasing dependency on data and the increasing complexity due to emerging dynamics and stochastics in the power systems. This requires a framework to facilitate the development, testing, and adoption of advanced solvers that can handle the data, dynamics and stochastics in the future power grid.

Framework Requirements

In the context of this project, the primary objective of the computational framework is to facilitate data exchange among different software modules in an application and to expose the problem to solvers and set up all necessary environments for pre- and post-solving, so solvers can be easily integrated and tested for any problems of interest. Below are the major requirements of this framework:

1. Efficient data handling (reading, curation, merging, format transformation).
2. Efficient scenario generation.
3. User-friendly result representation (e.g. visualization).
4. Data security control.
5. Problem representation in a numerical manner.
6. Problem representation in an equation-based manner.
7. Unified application programming interfaces (APIs) to solver integration.

Pathway with GOSS and GridPACK™

Figure 1 shows the general design of the computational framework, focusing on data and information exchange. Using the API, applications can seamlessly fetch data and information from multiple data sources or across different software modules. Instead of using hard-coded one-to-one interfaces, developers need only to use the unified interface when writing applications; compatibility and interoperability are ensured by the framework. The framework hides the specifics of data source management from developers. The framework allows software modules to communicate and exchange data using the APIs with a publish-subscribe mechanism.

This framework design can be implemented with a combination of GOSS and GridPACK, with extensions for additional functionality. GOSS would serve as a data management layer, responsible for bringing data from

data sources to computation, and from computation to visualization. GridPACK is a computing framework that is the actual place exposing a problem to solvers. Both GOSS and GridPACK are open source.

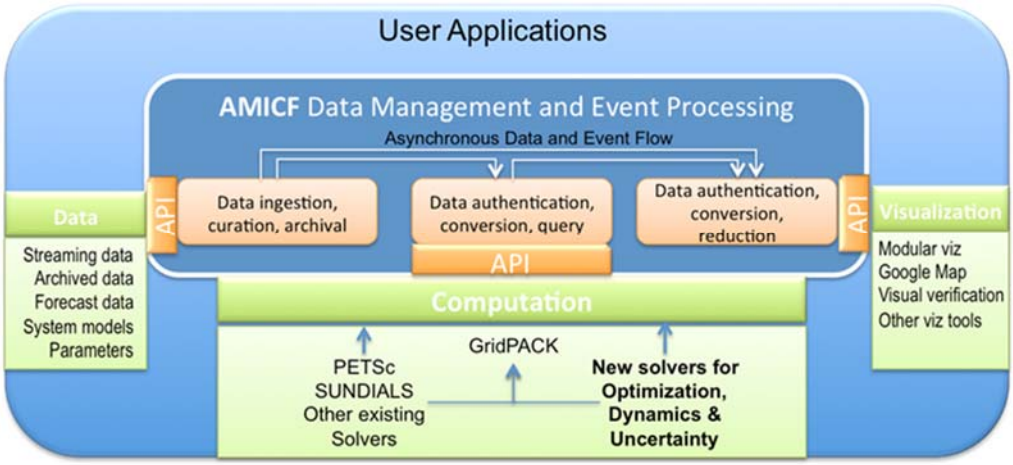

Figure 1. Conceptual design of the computational framework

GOSS (GridOPTICS™ Software System) https://github.com/GridOPTICS/GOSS

GOSS is an implementation of data management layer in the computational framework. It employs a publish-subscribe mechanism to facilitate data exchange between data sources and applications (See Figure 2).

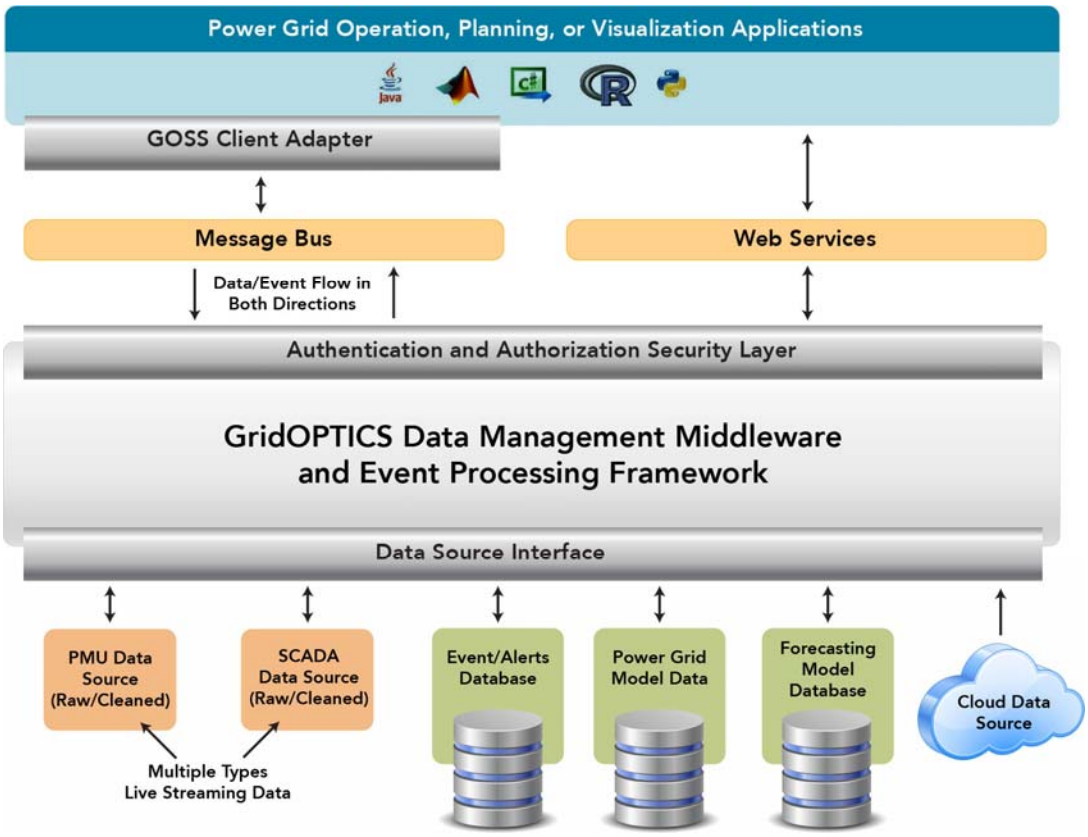

Figure 2. GOSS architecture

Per the requirements of the framework functions identified above, GOSS has already implemented the following capabilities:

1. Efficient data handling (reading, curation, merging, format transformation, event detection, data query and reduction).
2. Efficient scenario generation.
3. User-friendly result representation (e.g. visualization).
4. Data security control (data authentication and authorization).

GridPACK (GridOPTICS™ GridPACK) https://www.gridpack.org/

GridPACK has already been used for several power system applications including state estimation, transient simulation, and contingency analysis. Per the framework requirements, GridPACK has implemented the following capabilities:

1. Problem representation in a numerical manner.
2. Unified application programming interfaces (APIs) to solve integration.

GridPACK will be responsible for ingesting external files containing information on the power grid topology, devices and associated parameters, and any other information required for problem setup. This information will be transformed into the relevant equations and fed to the associated optimizing software via a standard interface that will be designed to support multiple different implementations using different solver and optimizer packages. The initial information will be read in using GridPACK parsing capabilities and used to set up a power grid network. Additional files may be used to add additional properties to the network. Once all information has been distributed to network objects and the network itself has been distributed over processors, GridPACK will generate the equations that are needed to define the optimization problem. These will be assumed to fall into three classes:

1) The objective function. This is the quantity that needs to be optimized, subject to constraints. It is assumed that this is some function of properties of individual network devices (e.g. cost per unit time of running the device)
2) Constraints on individual variables. If x is an optimization variable then this could be a constraint of the form $x \leq a$, where a is a parameter. Other inequalities are also possible. Constraints of the form $x = a$ would imply that x is just a parameter and does not need to be included in the optimization.
3) Constraints that are functions of several optimization variables. These could be of the form $x + y + \cdots \leq a$ where x,y,z etc. are associated with different devices.

Based on the properties of different devices on the grid and other additional information that may be included as part of the problem, GridPACK will generate the equations that will be passed to the optimization module. The current proposal is that GridPACK will formulate these equations as an internal expression object. The API will then convert these to the equivalent expression objects used by the optimization package. After completing the optimization, the optimizer package will then present results that must be distributed back to different network components.

Apart from determining whether this overall scheme is compatible with the proposed optimizer architecture, several specific tasks will be required in order to implement this scheme. These are

1) The data sources and input formats for setting up these problems must be identified. If standard formats for these types of problems are not available, then we will need to create something.
2) Based on available data, a procedure for generating the optimization equations needs to be developed. This is independent of any code.
3) GridPACK must develop software for parsing all relevant files and distributing information wherever it needs to go.
4) Software for generating the network equations and storing them in some kind of representation must be developed.
5) The GridPACK representation of the equations must be converted to the optimizer representation. This should be done behind the API so that the same GridPACK API can be used with multiple optimization packages (Figure 3).
6) Output from the optimization package needs to be extracted by GridPACK and either pushed back to network objects or sent to output directly.

Below are some additional questions that need to be addressed as part of this implementation:

1) The decomposition of the network in GridPACK will lead to a natural partitioning of the optimization variables and the resulting optimization equations. To what extent is this compatible with the input requirements of the optimization software and what adjustments will be required to resolve any discrepancies?
2) Are there equation representation objects in the optimization software that can be accessed by GridPACK to bypass the input deck? (We would like to avoid passing files between different parts of the program if at all possible).

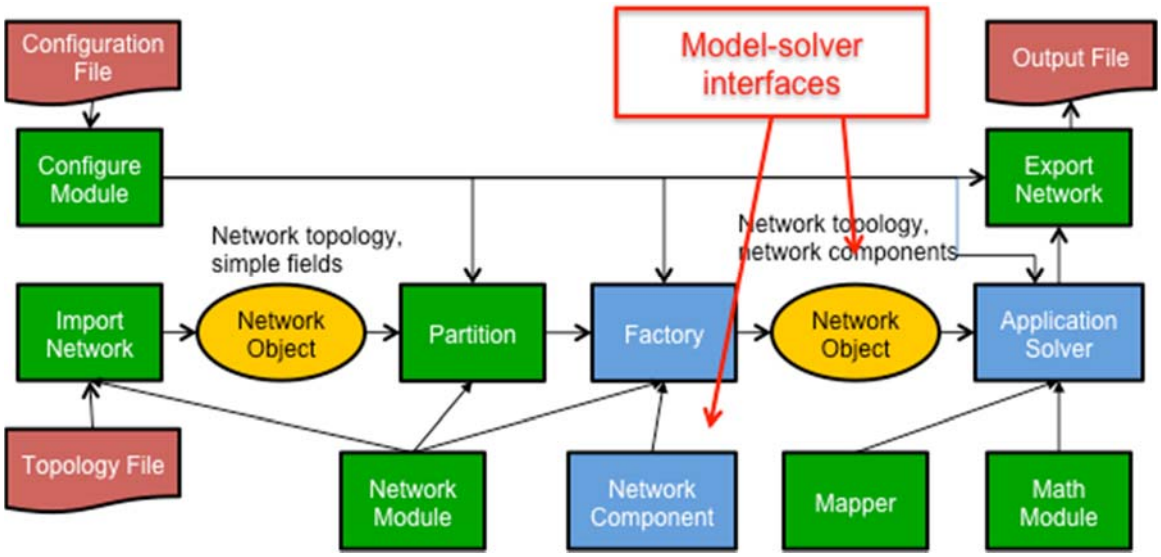

Figure 3. GridPACK structure

Extension

The combination of GOSS and GridPACK can meet most of the framework requirements. A major extension for GridPACK is the equation-based representation of the problems, which is important to link to StompJump, IDA, etc.