Real-Time Data Mediation for Synchrophasor Application Development Compliant with IEEE C37.118.2

Luigi Vanfretti[1,2], Iyad Al Khatib[3,4,5] and Muhammad Shoaib Almas[1]

[1] KTH Royal Institute of Technology, Sweden [2]Statnett SF, R&D, Norway [3] iITC, R&D, Stockholm, Sweden

[4]American University of Culture and Education (AUCE), Beirut, Lebanon [5]Politecnico di Milano (POLIMI), Milan, Italy

Abstract—**This paper presents the design and implementation of a IEEE C37.118-2-compliant real-time data mediator, namely BableFish (BF), that facilitates synchrophasor data-manipulation for smart grid applications. The mediator is scalable by virtue of its modular software architecture, and is capable of receiving concurrently synchrophasor data streams from either Phasor Measurement Units (PMUs) or Phasor Data Concentrators (PDCs). The different modules of the mediator and their functional capabilities are discussed. Furthermore, the performance of the mediator is analyzed through several statistical performance stress tests. The mediator facilitates prototyping of wide area power system monitoring and control applications using real-time hardware-in-the-loop simulation, as illustrated with examples in development at KTH SmarTS Lab. Finally, the technical challenges associated with implementation and integration of system-wide synchrophasor solutions for real-time synchrophasor-based applications is discussed.**

***Index Terms*— Synchrophasor Applications, Standard Implementation, IEEE C37.118, Real-Time Data**

I. INTRODUCTION

Power systems are becoming dependent on an increasing number of software applications and data connections, which are accompanied by complexity which can be largely estimated during implementation phases. These applications require data acquisition in real-time from new measurement units and data collection systems, and this brings in a complex set of challenges for software application designers [1]. Some of these challenges are: divergent product life cycles, varying scenarios of use, and increasing demand on real-time performance. In the very specific case of synchrophasor data collection in real-time, there could be large amounts of data gathered even in short time periods, which makes the search of data of interest very complicated, time consuming, and requiring high computational abilities in hardware systems. At the same time a specific application may not need all such data to be available.

To address the challenge above and give each user the data of interest, this article discusses the design and development of a real-time synchrophasor data mediator that: (i) connects to an arbitrary number of PMUs, PDCs, and other IEEE C37.118.2 [2] compliant devices, (ii) gives the user the ability to re-configure the choice of data to be used by a particular application out of all available data, and (iii) provides facilities to transmit the data-of-interest through any network (e.g. the Internet Cloud). The third characteristic allows to forward data to any remote unicast device or set of multicast devices, which can receive the sent data for presentation, monitoring, manipulation, and or processing it in other applications. The end user can choose the tools to receive data independently of platform, Operating System (OS), computer language, and geographical location.

The synchrophasor data mediator presented in this paper is named BableFish (BF), which receives real-time streams from PMUs/PDCs and allows engineers, researchers, or SW developers to implement PMU data applications. Thus, BF allows researcher/engineer/general user to do fast prototyping of new applications processing PMU measurements in their chosen environment. BF is scalable, modular (to guarantee easy future development), and capable of receiving several PMU data coming from either PMUs or PDCs.

Previous work [3] dealing with the development of a mediator (between PMUs or PDCs and an end user that may need to manipulate a subset of synchrophasor data) have focused on user interaction and validation of SW. This article focuses on the design and implementation aspects of such mediators.

The paper is organized as follows: Section II provides information about IEEE C37.118.2 compliance requirements. Different modules and functional capabilities of each are detailed in Section III. Performance stress test results are presented in Section IV, while integration of BF in SmarTS-Lab and developed power system monitoring and control applications are discussed in Section V. Finally in Section VI technical challenges associated with developing such a real-time data mediator are highlighted and conclusions are drawn.

II. COMPLIANCE WITH IEEE C37.118.2

The BF mediator has the ability to connect to multiple PMUs or PDCs by using an Internet connection, regardless of the underlying link layer used (which can be any kind of wireless or wired link). The application layer protocol IEEE C37.118.2 [2] is used for carrying configuration information and real-time data over TCP or UDP. Fig. 1 shows the subset of messages that are required from the IEEE C37.118.2

978-1-4799-1785-3/15/$31.00 ©2015 IEEE

Standard. To start the exchange of information between the BF (as a client) and the PDC/PMU, the first step is to establish a TCP connection between the two parties. Once the TCP connection is established, the BF can send a command frame to the PMU/PDC to either stop data transmission (Turn Off Data) or ask for a configuration frame.

A note worth mentioning is that in the IEEE C37.118.2 standard, there are two configuration frames ("Configuration Frame 1" and "Configuration Frame 2"), and for the mediator (BF) to be compliant with the standard, it is only necessary to send "Configuration Frame 2" from the client (mediator) to the PMU/PDC. Once the PMU/PDC receives the IP packet carrying a request from the BF client regarding the configuration frame, it processes this request and accordingly sends "Configuration Frame 2" to the client. Therefore, the mediator needs only to receive "Configuration Frame 2" and analyzing its data to be complaint with the standard.

After the mediator has all data needed for configuration, it sends another command to the PMU/PDC requesting it to send real-time data (Turn On Data). Consequently, when the PMU/PDC receives the Turn On Data request, it starts sending a continuous data stream to the mediator. Hence, the mediator needs to be able to receive IEEE C37.118.2 data frames and extract the data of every frame in real-time.

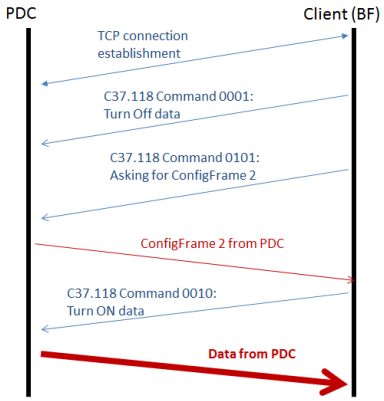

Fig. 1: Required messages to be compliant with the IEEE C37.118.2 protocol for connecting the BabelFish (BF) mediator to a PMU/PDC.

III. FUNCTIONAL DESCRIPTION OF THE REAL-TIME DATA MEDIATOR

The BabelFish mediator provides four main functionalities:

- M1: real-time reading from PMU/PDC
- M2: interfacing the reading module to LabView via ActiveX [5]
- LV M: LabView presentation layer and choice of data to use
- M3: sending chosen data to a remote host on the Internet in a simplified frame

Fig. 2 illustrates two major issues: (i) the interconnections between the BF and external hosts (PMU/PDC & the destination host), and (ii) the intra-connections between the BF internal modules (M1, M2, M3 & LabView LV-M).

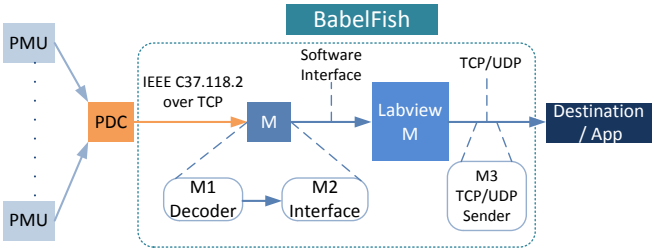

Fig 2: BablelFish modules M1, M2, M3, and "Labview (LV) M". M1 connects to the PDC. M2 interfaces between M1 and the Labview M, which is the presentation module showing the data on the screen. M3 gives the capability for the user to transmit the chosen data on the Internet to a remote destination (another user).

Module M1 and part of module M2 are in practice implemented in one module called M (Fig 2). M1 is responsible for connecting to the PDC/PMU(s) via TCP and is compliant with the application protocol IEEE C37.118.2 [2] as discussed in Section II. Module M2 is capable of reading real-time data from M1 and interfacing it with the LabView module "LabView M" (LV-M). M2 is a SW interface that consists of two parts: (1) one in M (as discussed earlier) and (2) the other in the LV-M. Since the mediator deals with real-time data, performance parameters such as delay and packet loss are essential to study for choosing the most suitable interface method for module M2. Experimental tests were designed and performed to select the best interfacing scheme. These experiments and relevant results are discussed in section IV.

The LV-M allows the user to graphically choose the data of interest in order to filter data streams in real-time. Comparing this SW design with state-of-the-art applications (from the points of view of use of HW, parallel SW modules, and time spent on preparing and integrating tests), we note that the proposed design allows performing real-time monitoring and analysis with: (a) faster reading, (b) less computational power from the processors, and (c) smaller memory sizes. Such advantages seen from the beginning of the design process help in providing scalability (i.e. adding more PMUs and PDCs to be handled per machine or mediator). This design concept opens the possibility to use smaller electronic boards to run the same mediator, therefore, decreasing cost and allowing for the use of PMU data in diverse applications.

A. Architecture and choices for modularity

The BF modules in Fig. 2, are the same ones discussed in this section (shown in detail in Fig. 3); however, Fig. 2 presents them from a SW point of view and considers modularity while Fig. 3 presents the SW functional architecture. For faster performance while reading real-time data, module M1 was implemented in Visual C++. The C++ code in M1 includes an executable and two Dynamic-Link Libraries (DLLs), which are responsible for *connecting to*, *sending frames to*, and *receiving frames/data from* the PDC. At the same time, the executable of M1 contains part of M2 (as shown in Fig. 3) and it utilizes ActiveX to send and receive data to/from LV-M. This requires module M2 to be

implemented in two different computer languages: (i) the first part in VC++ (inside M1 as discussed above) and another part in a LabView programming language called Virtual Instruments (VI) [4] for reading ActiveX variables. Finally, the LV-M gets the user-chosen data (from the user GUI) in order to present this data in real-time. The user may want to send this chosen data to a remote destination via module M3, which is also implemented in LabView VI. For the specific implementation discussed in this paper, M3 utilizes UDP. The use of TCP is planned for a later version of the software.

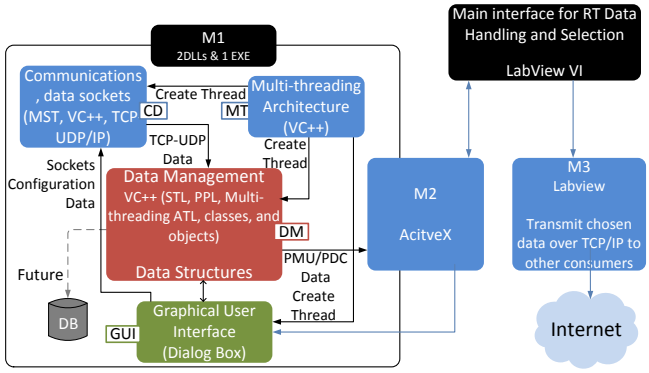

Fig. 3: Modular software architecture of BabelFish.

B. Implementation of modules

The implementation of Module M1 shown in Fig. 3 divides the functionalities of the module to four sub-modules that are fully within M1 and one sub-module (part of M2). The sub-modules are: (1) the Communication Data (CD) that is responsible of communicating with the PMU/PDC, (2) the Multi-Threading (MT) sub-module that takes care of dividing the tasks on different threads so that parallel functions are executed simultaneously (such as reading real-time data from the PDC while at the same time analyzing received data frames and sending the data to ActiveX), (3) the Data Management (DM) sub-module that takes care of the data received from the PDC and sends it to M2, (4) the GUI sub-module (see Fig. 4) that initializes the connection with the PDC and asks for "Configuration Frame 2", and (5) the part of M2 that uses ActiveX for communication with LabView.

The GUI is an executable entitled PDC_Process_Dlg (see Fig. 4). The other sub-modules are implemented in practice in two DLLs that deal with contacting the IEEE C37.118.2 compliant device and filtering information. Therefore, when PDC_Process_Dlg is executed, it calls upon functions from the two DLLs, which are related to the sub-modules presented in M1 as shown in Fig. 3. The second part of M2 is integrated together with the LabView module (LV-M) in one VI, but the M2 part is only responsible for ActiveX data reading. The LabView module includes the presentation part of the SW, and it is shown in Fig. 5 and Fig. 6. After having connected with Module 1 (Fig. 4) using VC++, the LabView VI (Fig. 5) shows the PMUs connected to the PDC, and it allows the user to choose which PMU data are of interest (down to the detail of choosing either the Magnitude or the Angle of the synchrophasor of interest). The chosen data is then displayed in a table in the lower left corner of the LV-M (see Fig. 5).

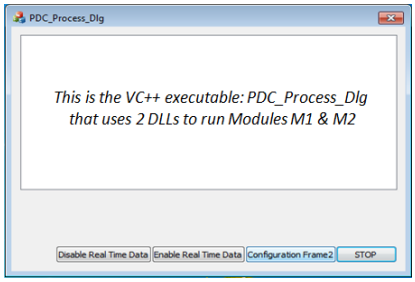

Fig. 4: Visual Studio executable Dialogue window to connect the mediator to the PDC, entitled "PDC_Process_Dlg". The user clicks on the "Configuration Frame2" button in order to start the event of sending a command asking for ConfigFrame 2 of the IEEE C37.118.2 from the PDC.

The user can either save a configuration or load an older configuration by using the "Save" and "Open" buttons. This would save time for the user, who may be interested in specific synchrophasor values for a particular application. Then, by pressing the "Play" button, the VC++ code (Module M1) knows by an event sent from the module (LV- M) that it has to filter only the chosen data and accordingly send (via M2 using ActiveX) the filtered real-time data values, which get displayed in real-time also in the GUI lower tab, as show in Fig. 5. In addition, the user can choose to send the real-time filtered data to a remote destination by going to the lower tab entitled "Output Cluster" and pressing on the "Start" button (see Fig. 6). Consequently, the data will be sent over UDP to a chosen host. The configuration of the remote host is also chosen by the user in the upper right tab entitled "Transport Info". Fig. 7 shows the demonstration of a set of chosen PDC data received in real-time at a remote host running a UDP receiver. With this implementation, the users at both, the BF side and remote destination, can use the data of interest in real-time.

IV. Performance

The mediation scheme presented requires Quality of Service (QoS) checks since it deals with real-time data. The SW presented here was designed after QoS tests were made for three different scenarios to interact between VC++ and LabView. In order to maintain the interaction between these two environments without delay, a relatively fast common interface is needed. Three different interfaces were evaluated: text-based or Data Base (DB), streaming, and ActiveX. The results from 100 statistical performance tests are presented in Table 1. These results suggest that ActiveX is the most suitable environment for real-time implementations in Windows 7 OS since its delays are significantly different and it shows no packet loss.

The performance of the real-time reading module in VC++ depends on the computational power of the HW used. Stress tests could not show the limit. The tests were performed on a PC with Intel Core i5 CPU with 4GB RAM using Windows 7 OS (only 3.7 GB RAM effectively available). For the stress test, a PDC with more than 6 PMUs, each with more than 4 parameters to transmit was used; the performance was not affected. The total end-to-end delay was in the millisecond range, and no glitches were observed out of 100 statistical tests. Hence an open issue remains: to have as many PMUs connected to check the limit on performance.

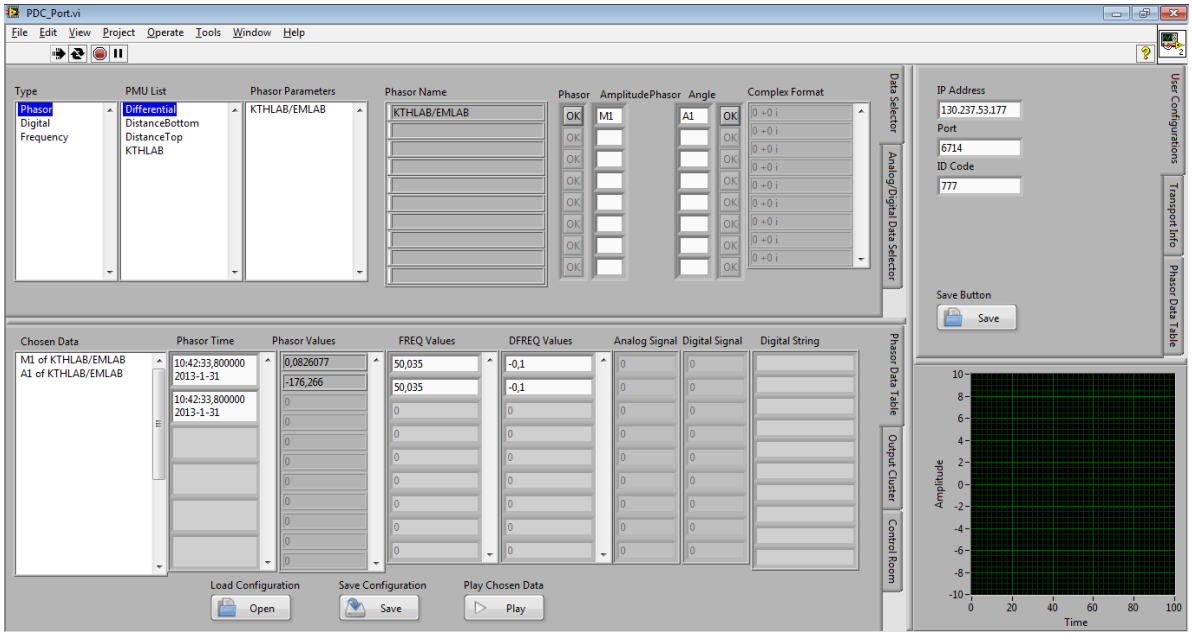

Fig. 5: LabView Module (LV-M) for choosing data of interest by the user and displaying the real-time stream of chosen data. The top tab shows the list of PMUs that are connected to the PDC, the synchrophasors per PMU, and the "OK" buttons to choose the Magnitude(s) and Angle(s) of interest from each synchrophasor. The part on the right shows the network configuration tabs, and the one presented is for the IP address, Port, & ID Code of the PMU/PDC. The lower tab on the left shows the Chosen Data. When the user presses Play, the synchrophasor real-time information is displayed continuously: Phasor Time Stamp, Phasor Values, FREQ, DFREQ, Analogs, and Digitals.

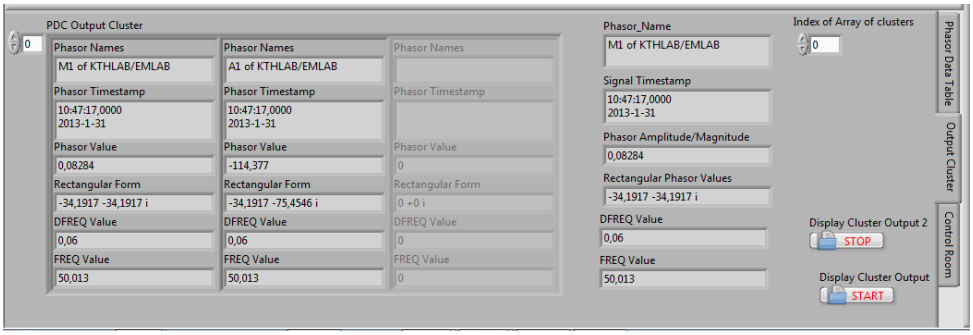

Fig. 6: Labview Module presenting the chosen data in the lower part in the tab entitled "Output Cluster", where each Phasor data set is clustered in one column and ready to be sent to a remote destination when the "START" button is pressed (see sent data in Fig. 7).

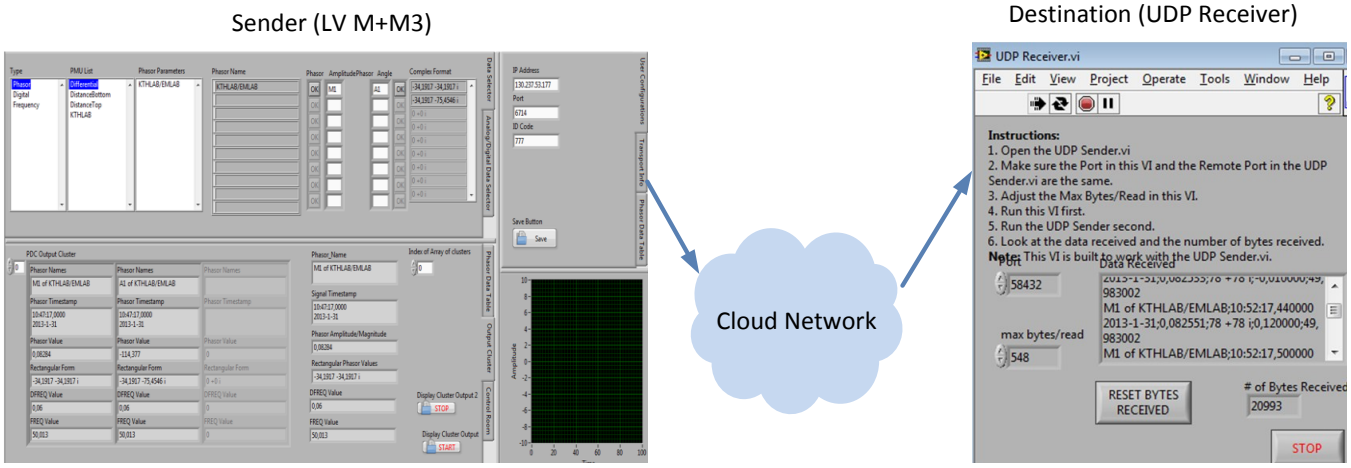

Fig. 7: Sending a set of chosen PDC data from the LV Module to a remote destination over the Internet. The destination shows a UDP receiver waiting for the clustered data (from Fig. 6) to be read at port 58432 (chosen by the users via an agreement on both sides) and the data is presented in the scroll down box.

TABLE I
QOS TESTS FOR 3 DIFFERENT SCHEMES (DELAY VALUES ARE IN ms)

Parameter	Text Based	Streaming	ActiveX
Average Delay	2.54	1.49	0.2
Min. Delay	0	0	0
Max. Delay	8	13	1
Std. Deviation	1.17	1.34	0.40

V. REAL-TIME EXECUTION OF BABELFISH

The developed BF is integrated in SmarTS-Lab [6] to provide effortless implementation of synchrophasor based monitoring, protection and control applications. The workflow for real-time hardware-in-the-loop execution of BF is shown in Fig. 8. The three phase voltage and current signals of the desired buses of a power system model are accessed from Opal-RT's eMEGAsim Real-Time simulator [7] and are fed to the PMUs. These PMU streams are time aligned and concentrated by the Phasor Data Concetrator (PDC) and are received by BF executing in a workstation. The user selected raw synchrophasor data through BF is further transmitted to different power system monitoring, and control applications using UDP thus facilitating platform independency for developing these synchrphasor based applications. The bottom half of Fig. 8 shows the prototype applications being developed using BF where; (i) is the simple UDP client on either a remote workstation or smart phone / tablet to receive user selected synchrophasor data for visualizing power system trend, (ii) is an external controller based on National Instruments Compact Reconfigurable I/O Controllers (NI-cRIO) receiving selected raw synchrophasors from BF as UDP and utilizing these measurements in real-time for power oscillation damping algorithm, and (iii) shows a power system monitoring application developed in LabView to visualize frequency, voltage/current magnitudes and phase angles as selected by user in BF. A more detailed application example is presented in [8].

VI. CONSLUSIONS AND DISCUSSION ON IEEE C37.118.2 TRANSFORMATION TO REAL-TIME

Transforming a textual written standard for a protocol into a coded application within a networking environment, while considering the large amount of growing networking protocols (and versions of protocols) and data link types (e.g. Wireless LANs, 2.5G, 3G, 4G, wired links, etc.), a set of challenging issues that could hinder the process of a suitable design of solution (and modules) naturally emerges. The major challenge was to be able to account within the design process for all possible combinations of Operating System differences, platforms, TCP versions, TCP/IP suite implementations, graphical design issues and their use of available HW resources, parallelization of the different modules and what comes before/after what while the protocol is interacting with clients and servers. These issues were learnt over the process of implementation, and one learning from this work is that they better be taken into consideration from the beginning of the solution design. In this work a co-design process considered the issues mentioned above and specific tests for every issue of interest were designed and performed in order to make choices on HW and SW. For instance, the results in Table 1 lead to the choice of ActiveX as an interaction module due to the fact that the solution was

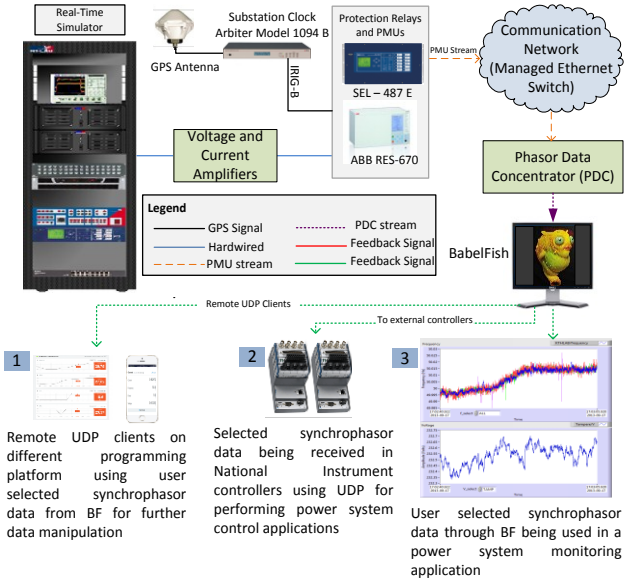

Fig. 8: Utilization of BF in SmarTS-Lab to develop platform independent wide area monitoring and control applications.

built for real-time utilization, i.e. delay is a sensitive parameter. In this regard, one important result is the threshold on the delay that the solution can handle. This delay is a function of the HW specifications (presented in Section IV), the SW parallel modules, and the network environment, and the interface modules (e.g. text, streaming, or ActiveX). The result is that the only critical parameter is the end-to-end delay. Hence, during the design and implementation phase, it was important to make integral tests for the delay of each module, as shown in Table 1. However, after the whole implementation is finalized, a check is necessary to determine if the delay from the PDC to the BF and to the remote user are within real-time limits.

VII. REFERENCES

[1] E. Santacana, G. Rackliffe, L. Tang, and F. Xiaoming "Getting Smart", IEEE Power and Energy Magazine, Vol. 8, No. 2, pp. 41 – 48, April 2010.

[2] IEEE C37.118.2-2011, IEEE Standard for Synchrophasor Data Transfer for Power Systems, IEEE Power and Energy Society, Dec. 2011

[3] L. Vanfretti, V. H. Aarstrand, M. S. Almas, V. S. Perić and J. O. Gjerde , "A Software Development Toolkit for Real-Time Synchrophasor Applications", IEEE Powertech 2013, Grenoble, France, June 2013

[4] National Instruments, "Labview-System Design Software", documentation available online: http://www.ni.com/labview/

[5] Microsoft, "Component Object Model – ActiveX", documentation available online: http://www.microsoft.com/com/default.mspx

[6] M.S. Almas, M. Baudette, L. Vanfretti, S. Løvlund and J.O. Gjerde, "Synchrophasor Network, Laboratory and Software Applications developed in the STRONg^{2}rid project", IEEE PES General Meeting, Washington DC, USA, July 2014

[7] Opal-RT, "eMEGAsim PowerGrid Real-Time Digital Hardware in the Loop Simulator," available on-line: http://www.opal-rt.com/.

[8] L. Vanfretti, M. Baudette, I. Al-Khatib, M. S. Almas, and J. Gjerde, "Testing and Validation of a Fast Real-Time Oscillation Detection PMU-Based Application for Wind-Farm Monitoring", Black Sea Conference on Communications and Networking, Georgia, July 2013