LabVIEW INTERFACE FOR MADOCA II WITH KEY-VALUE STORES IN MESSAGES

T. Matsumoto#, Y. Furukawa, Y. Hamada, T. Matsushita, JASRI/SPring-8, Hyogo, 679-5198, Japan

Abstract

As the next generation of the Message and Database-oriented Control Architecture (MADOCA), MADOCA II is a message-driven distributed control framework that is more accommodating, e.g., it allows control on Windows operating system as well as messages consisting of data of variable [1]. A prototype of MADOCA II was developed in the Laboratory Virtual Instrument Engineering Workbench (LabVIEW) interface in 2013 and implemented on a control system in SPring-8 [2]. However, it was recognized that the interface should be easy-to-use in order to be able to implement MADOCA II in a wide variety of LabVIEW applications, especially in synchrotron radiation experiments. In this paper, we propose a redesigned LabVIEW interface to respond to this challenge. In this interface, messaging is managed through a unified method with virtual instruments (VIs) using key-value stores. As a result, applications for the client program and the equipment management server can be easily constructed. The VIs are based on a dynamic link library (DLL) developed in C++. The interface is easily upgraded by replacing the DLL, which was also applied to the Python interface.

INTRODUCTION

The Message and Database-oriented Control Architecture (MADOCA)-II is the next generation of MADOCA. It was successfully implemented in accelerator controls in the SPring-8 and the SPring-8 Angstrom Compact Free Electron Laser (SACLA) data acquisition (DAQ) system in 2013, as reported at the last International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS) [1]. Like its predecessor [3], MADOCA II is based on a message-oriented control scheme. A message is composed of a text in subject/verb/object/complement (S/V/O/C) syntax. In this context, “S” denotes the program that is automatically defined by the framework and “V” denotes the action of a command, for which “put” or “get” is primarily used. “O” is an object name, which identifies the target of the message, and “C” is an action parameter. Figure 1 shows the structure of MADOCA II messaging. A graphical user interface (GUI) sends a message with “put/camera/init” as V/O/C (S is abbreviated as described), and an equipment manager (EM) of device control responds to the message in the S/V/O/C format. If the initialization of the camera is successful, C is set to “ok” in the response message. Thus, this messaging scheme is easy to understand.

Moreover, MADOCA II is more flexible than MADOCA. In MADOCA II, data strings of varying length, such image data, can be attached to a message, and the Windows environment can be used for device control. These features were utilized in control applications in SPring-8.

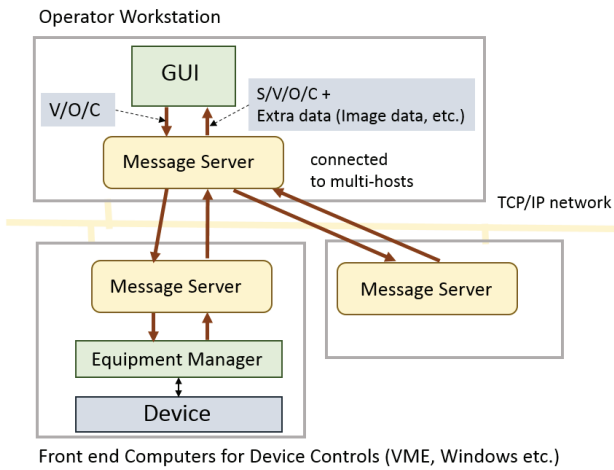

Figure 1: The structure of MADOCA II messaging. Messages are exchanged between a GUI and an equipment manager (EM) thorough a Message Server (MS) in each host.

A motivation underlying the development of MADOCA II was the ability to use the Laboratory Virtual Instrument Engineering Workbench (LabVIEW) interface by using the above flexibilities because LabVIEW is used for some parts in the accelerator control and many applications in synchrotron radiation experiments. If these LabVIEW applications are updated to contain an interface for MADOCA II, we can unify its control procedures in order to save on management costs.

In 2013, we developed a prototype of the LabVIEW interface for MADOCA II, and applied it to a beam position monitor (BPM) with NI PXI-5922 digitizers [2]. In the application, waveform data was attached to a message. A total of 5,000 samples of the waveform could be exchanged between a client application and an EM of the BPM within one second, and the BPM application functioned stably.

We intend to apply the MADOCA II LabVIEW interface to other LabVIEW applications. However, we realize that the interface should be revised for ease of use and better maintainability.

#matumot@spring8.or.jp

ISBN 978-3-95450-148-9

The prototype of the MADOCA II LabVIEW interface was implemented by porting the algorithm of MADOCA II with LabVIEW, excluding LabVIEW-zmq binding [4] and Message Pack for LabVIEW [5]. We occasionally needed to control the version of ZeroMQ and MessagePack used in MADOCA II. However, this was difficult because we did not maintain LabVIEW-zmq binding and Message Pack for LabVIEW. Furthermore, updating the library required elaborate work because we needed to care both for the library for C++ and LabVIEW. We redesigned the LabVIEW interface for MADOCA II to solve these problems. In this paper, we describe this redesigned MADOCA II LabVIEW interface and its applications.

DESIGN OF MADOCA II LabVIEW INTERFACE

We redesigned the MADOCA II LabVIEW interface for better maintainability and ease of use. With regard to maintainability, we developed the LabVIEW interface using a Dynamic Link Library (DLL). The virtual instruments (VIs) call the DLL to use functions in MADOCA II. Since the DLL is constructed with the MADOCA II library in C++, the algorithm for MADOCA II can be shared between LabVIEW and other applications in C++. The MADOCA II library can be easily upgraded by replacing the DLL. We can also manage the versions in ZeroMQ and the Message Pack at the same time.

With regard to ease of use of the interface, we decomposed the messaging steps in MADOCA II and implemented each step through a corresponding VI. We can easily construct an application by arranging these VIs. To simplify the interface, we introduced VIs with key-value store to manage information contained in the messages. We prepared an internal buffer in the DLL. When we build message information using a VI, the information is stored in the internal buffer. The contents can be obtained from other VIs by fetching the information in the buffer.

In MADOCA II, we can treat messages containing data of variable length, such as image data. This feature is useful, but developing an easy-to-use interface for this is challenging. Using the key-value stores, we can develop a unified method to access this information, and it becomes very easy to handle various kinds of data with this messaging system.

The DLL used in the LabVIEW was applied using C++ and Python as well. MADOCA II applications can be easily built from other languages in a similar manner. The Python interface was prepared by using ctypes modules. MADOCA II can be easily used through the script language.

The details of the LabVIEW interface are shown below.

Implementation of VIs for Applications

We now describe the implementation of VIs for a client application and an equipment management (EM) application. Figure 2 shows a flowchart of a client application. The steps involved in messaging are shown with corresponding icons for the VIs. “MS2_OPEN” first connects to an MS. At this time, an internal buffer is created, followed by a handle to share information with other VIs. Following this, the internal buffer is initialized with “MSG_BUILD_INIT.” The message is then built using “MSG_BUILD_INF.” For example, we can set the key as “VOC” and the value to “put/camera/init.”

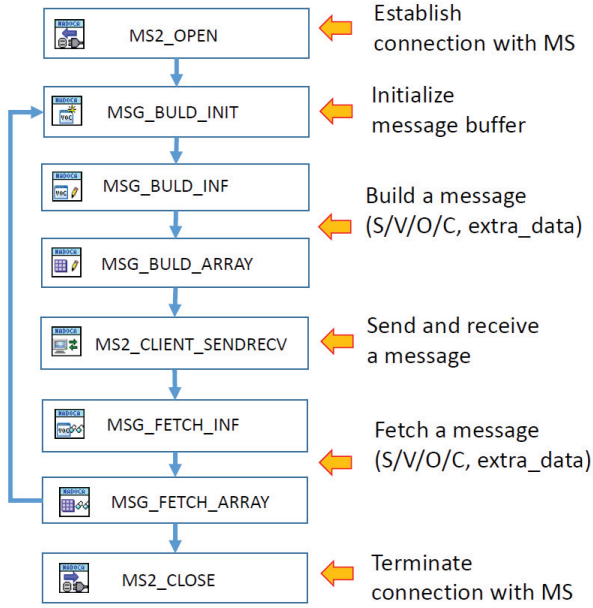

Figure 2: Flowchart of a client application in MADOCA II.

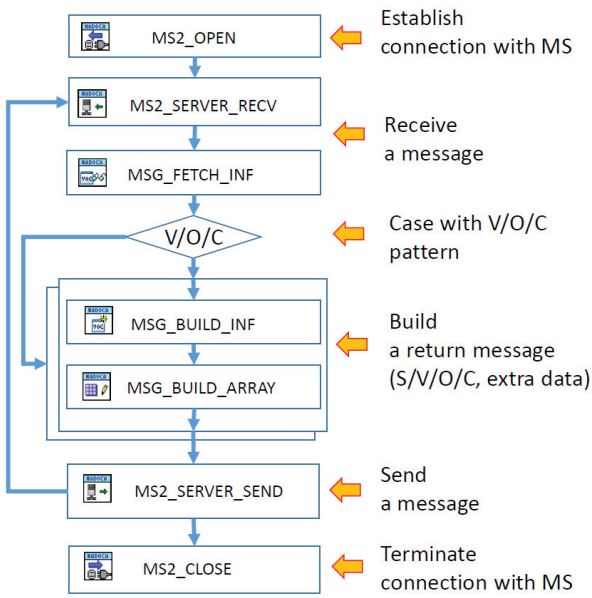

Figure 3: Flowchart of an equipment manager server application in MADOCA II.

ISBN 978-3-95450-148-9

The extra data attached to the message can be built using "MSG_BUILD_ARRAY." For example, we set the key as "image_data" and a value for the relevant array data. Various data types can be specified, such as integer and float. Following the construction of the message, we send and receive messages using "MS2_CLIENT_SENDRECV." The received messages are stored in the internal buffer in the DLL. "MSG_FETCH_INF" and "MSG_FETCH_ARRAY" can then be used to fetch required information. Finally, "MS2_CLOSE" terminates the connection with the MS.

The case of a flowchart for an EM application is shown in Figure 3. In case of an EM, "MS2_OPEN" is also used to form a connection with an MS, but the object list managed in the EM is set at the same time. "MS2_SERVER_RECV" is then used to receive messages from client applications. Following the receipt of a message, information is stored in a buffer in the DLL and fetched through "MSG_FETCH_INF." The response of an EM is determined by checking the V/O/C pattern of the relevant message, and the reply is constructed using "MSG_BUILD_INF" and "MSG_BUILD_ARRAY." Finally, "MS2_SERVER_SEND" sends the response to client applications.

Thus, we can easily construct LabVIEW programs for client applications and EM applications.

VI Components of MADOCA II LabVIEW

A total of 23 VIs were implemented for the LabVIEW interface of MADOCA II. Some of these VIs are shown in Figure 2 and Figure 3. Icons were prepared for each VI in order to easily identify its use. We also prepared VIs to treat various types of extra data, including hierarchical structures.

An error handle was attached to each VI of the interface in order to identify the error. Once we find the error, the error content can be obtained through text from other VIs to understand its cause during message routing in MADOCA II.

Robustness against Control Troubles

We designed the MADOCA II LabVIEW interface to be robust against control issues for an easy-to-use interface. In MADOCA II, messages are exchanged by using object information registered in the MSs. Object information was registered to an MS when an EM started. However, if the EM terminated unexpectedly, we often found that object information was not cleared from the relevant MS. In this case, we could not restart the EM unless we manually cleared object information from the MS, since a duplication of the object in an MS is forbidden. To solve the problem, we first cleared object information related to the application, and then restart the EM. Thus, we avoided problems related to use in MADOCA II.

Available Data Format in MADOCA II

In MADOCA II, extra data such as image and waveform data can be attached to a message. The extra data can be built and fetched with key-value stores in the MADOCA II LabVIEW interface. However, it is important to have unified data format to share information with different applications. Therefore, we defined data formats for several data types, such as image, waveform, camera controls, etc. An example of the data format for image data is shown in Table 1. In case of image data, the data type was defined by the key "image_data_type," and image size was defined by "image_width," "image_height," and "image_depth." The data type of the image was defined by "image_num_type," and the image array data was defined by "image_data." Data format was also applied to the MADOCA II library with C++ and used in a control application for image transfer in a two-dimensional interferometer in SPring-8 [6].

Table 1: An Example of Data Format for Image Data

Key	Data type	Value
image_data_type	string	"MONO","RGB","RGBA"
image_width	int32_t	
image_height	int32_t	
image_depth	int32_t	
image_num_type	string	"uint8_t","uint16_t","uint32_t","uint64_t","int16_t","int32_t","int64_t","float","double"
image_data	defined by image_num_type	
image_pixel_order	string	"lefttop," "leftbottom" ("lefttop" if not defined)

ISBN 978-3-95450-148-9

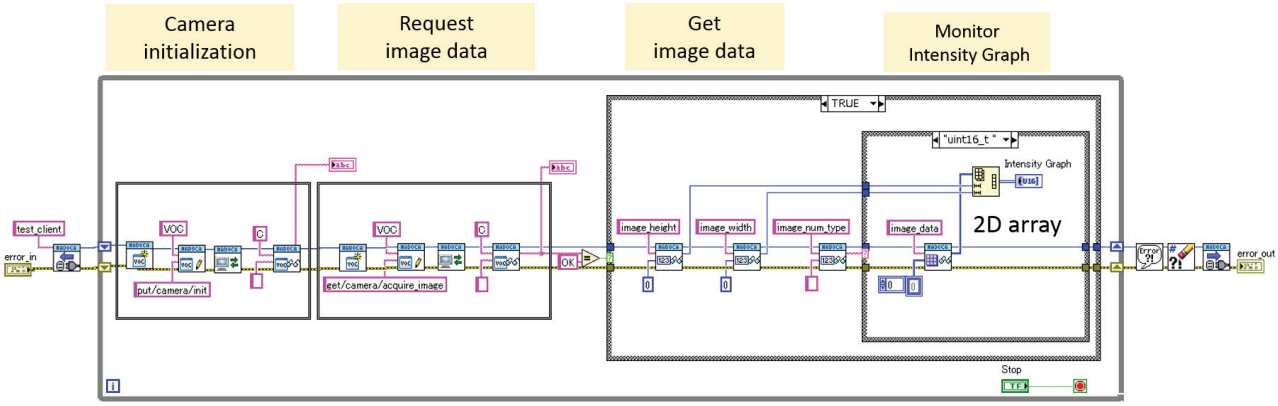

Figure 4: An example of a client program to monitor a camera image with MADOCA II LabVIEW interface. The camera images were remotely obtained through MADOCA II messaging from a camera server.

CAMERA IMAGE VIEWER WITH MADOCA II LabVIEW INTERFACE

Figure 4 shows an example of a client application for a camera image viewer using the MADOCA II LabVIEW interface. The application performs remote control of a camera and monitors the intensity graph of the resulting image. In the application, the camera is initialized with a "put/camera/init" command, and image data is requested with a "get/camera/aquire_image" command. The obtained data is stored in an internal buffer in the DLL and fetched by specifying the corresponding keys: "image_height," "image_width," "image_num_type," and "image_data." Then, the obtained data array is monitored as an intensity graph with LabVIEW. Such an application can be easily programmed with the MADOCA II LabVIEW interface.

For synchrotron radiation experiments, we tested an application using a high-sensitivity camera, the Hamamatsu ORCA-Flash 4.0. We prepared a camera server with an EM for ORCA-Flash and monitored the camera image from another client PC with the same application as shown in Figure 4. The PCs were connected to a 1 GbE network switch under a local network. A camera image of 4 MP could be monitored with a few Hz rate.

In the MADOCA II LabVIEW interface, we used an internal buffer in the DLL for ease of use with key-value stores. The control application may be not suitable for quick control because we the internal buffer is mediated during messaging. However, we can apply the interface for slow control applications, such as monitoring and device controls in SPring-8.

SUMMARY

We redesigned MADOCA II LabVIEW interface for ease of use and better maintainability. We applied key-value stores to VIs to easily manage various data in messaging with a unified method. The LabVIEW interface is based on the DLL and can be easily upgraded by replacing it. The DLL is also applicable to other languages, such as C++ and Python. With the redesigned interface, MADOCA II can be easily applied to LabVIEW and enables rapid programming. We will apply the redesigned LabVIEW interface to several control applications in SPring-8 in future research. We are preparing a LabVIEW application with an interface for a monitoring system for the NewSUBARU accelerator. We also plan to add functions to the LabVIEW interface for greater flexibility in control applications.

ACKNOWLEDGEMENT

We are grateful to Chuo Electric Works Ltd. for developing the LabVIEW interface for MADOCA II.

REFERENCES

[1] T. Matsumoto et al., "Next-generation MADOCA for SPring-8 control framework", Proceedings of ICALEPCS 2013, San Francisco, USA, (2013) p. 944.
[2] Y. Furukawa et al., "MADOCA II Interface for LabVIEW", Proceedings of ICALEPCS 2013, San Francisco, California, USA, (2013) p. 410.
[3] R.Tanaka et al., "The first operation of control system at the SPring-8 storage ring", Proceedings of ICALEPCS'97, Beijing, China (1997) p.1.
[4] http://zermq.org/
[5] http://msgpack.org/
[6] A. Kiyomichi et al., "Development of MicroTCA-based image processing system at SPring-8", Proceedings of ICALEPCS 2013, San Francisco, California, USA, (2013) p. 78.

ISBN 978-3-95450-148-9