Communication in a LabVIEW Based Holonic Controller

D.M. Masendeke[1], A.H. Basson[1]
[1]Dept of Mechanical and Mechatronic Engineering, Stellenbosch University, South Africa

Abstract
This paper presents a communication approach for a holonic controller developed in LabVIEW. The controller is aimed at station that forms part of a reconfigurable assembly cell. The objective of the research was to evaluate the extent to which LabVIEW facilitates reconfigurability in this context. The holons were implemented by using LabVIEW's producer/consumer (to achieve asynchronous inter- and intra-holon communication) and state machine architectures. The paper discusses some implementation considerations. The paper shows that LabVIEW offers some attractive facilities for lower levels of reconfigurable control systems.

Keywords
LabVIEW; Reconfigurable manufacturing system; Holonic control system

1 INTRODUCTION

Laboratory Virtual Instrumentation Engineering Workbench (LabVIEW) is a platform and development environment for a visual programming language from National Instruments (NI). This paper evaluates LabVIEW as a platform for the control for a subsystem of a reconfigurable manufacturing system (RMS).

The motivation for considering LabVIEW is that other control approaches, such as agent-based control (which have been used in most RMS research) and IEC 61499 function blocks, have not found favour with industry. IEC 61131-3 languages, ubiquitous in manufacturing control, on the other hand, have not found application in RMS control systems, although Hoffmann [1][2] has presented some work in this regard. LabVIEW is considered here because it has been used with success for test equipment in manufacturing environments and also for major scientific projects, such as the control of the South African Large Telescope (SALT). LabVIEW runs on mature and reliable hardware and there is a substantial user base, in particular at universities and research institutions. People that are not skilled in high-level programming can use LabVIEW and this is significant in manufacturing environments, that the authors are familiar with, where technicians are rarely skilled in object orientated programming and even less so in agent-based programming,.

LabVIEW is commonly used for data acquisition, instrument control and industrial automation [3]. LabVIEW programs or subroutines are called Virtual Instruments (VIs), because their appearance and operation imitate physical instruments such as oscilloscopes and multi-meters [3]. Each VI has three components: a block diagram, a front panel and a connector pane. The connector pane is used to represent the VI in the block diagrams of other calling VIs. The front panel is built with controls and indicators, which are the interactive input and output terminals of the VI, respectively [4]. Therefore, controls and indicators on the front panel allow an operator to input data into or extract data from a running VI. It is worthwhile to note that the front panel also serves as a programmatic interface. After the front panel has been built, graphical representation is used to add code to control the front panel objects. The block diagram contains this graphical source code [4].

An RMS is a responsive manufacturing system, which includes that its production capacity is adjustable to fluctuations in product demand and its functionality is adaptable to new products [5]. RMSs are ideally flexible (able to manufacture a limited family of products in a given configuration), convertible (amenable to adding or removing subsystems in a reasonably short time), scalable (able to adapt to changes in production volumes required by the market), modular (in its hardware and control), integrable (the interfaces between modules are amenable to easy integration) and diagnosable (able to detect fault conditions). These key characteristics help achieve the desired reduction in reconfiguration time and cost [5]. The goal in implementing an RMS is to be able to cope with rapid, unpredictable changes in production requirements through reconfiguration [6].

RMS control systems are widely implemented using holonic control architectures since these architectures offer good modularity and integrability in the control system. The relevant modules in the control system are also directly associated with hardware modules, thereby simplifying the mapping of the hardware to reconfiguration to the control system. Product-Resource-Order-Staff Architecture (PROSA) [7], one of the best known reference architectures for holonic control, was chosen for the research presented here.

The research presented in this paper contributes to an evaluation of LabVIEW's ability to facilitate reconfigurability within the context of the controller

for a station inside a manufacturing cell. The evaluation used a rivet feeder station as a case study. This station is part of the RMS cell being developed at the Mechatronics, Automation and Design Research Group at Stellenbosch University. The rivet feeder station includes a vibratory bowl feeder, a singulation device, a pick-n-place mechanism and an XYZ positioning table (Figure 1). More details about the case study are given in [8].

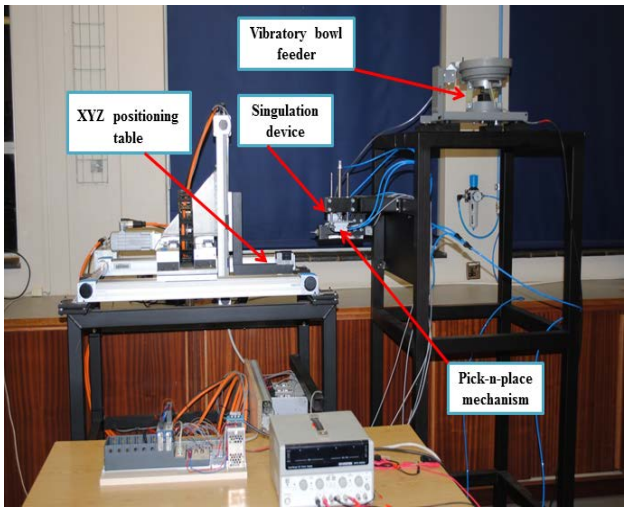

Figure 1 - Rivet feeder station hardware

2 CONTROLLER ARCHITECTURE

2.1 Station controller

PROSA was selected as reference architecture since reconfigurability is enhanced by PROSA's high degree of self-similarity, which reduces the complexity to integrate new components and enables easy reconfiguration of the system [7].

In PROSA, a product holon holds the process and product knowledge required for the correct manufacturing of a particular product type. The product holon acts as an information server to the other holons in the holonic manufacturing system [7]. A resource holon contains a physical part (a production resource) of the manufacturing system, as well as an information processing part that controls the resource. A resource holon offers production capacity and functionality to the surrounding holons and can be seen as an abstraction of the production means such as a factory, a shop, machines, furnaces, conveyors, pipelines, pallets, etc [7]. An order holon represents a task in the manufacturing system. It is responsible for performing the assigned work correctly and on time [7]. The staff holons assist the aforementioned basic holons in performing their work and can reduce the work load and work complexity of the basic holons, by providing them with expert knowledge [7].

Figure 2 shows the architecture of the LabVIEW based controller, consisting of six holons, i.e. coms holon, request manager holon, order holon, product holon manager, pick-n-place holon and XYZ table holon. The coms holon and the request manager holon are staff holons, while the XYZ table holon and the pick-n-place holon are resource holons. The station controller, in its present form, makes provision for only one order holon and all the product holons are represented by a single product holon manager.

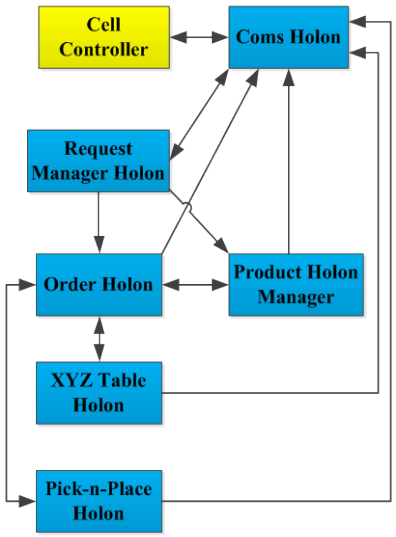

Figure 2 - Rivet feeder station control architecture

2.2 Common holon internal architecture

Each holon must be able to run independently from other holons. Therefore each holon was implemented in LabVIEW using a while-loop containing all its functions. This while-loop terminates only when the command to shut down is communicated through a notifier.

The intelligent decision-making algorithms of the holons were implemented using LabVIEW's state machine architecture [9]. The state machine associated with each holon is implemented as a case structure inside the holon's main while-loop. The holon's main while-loop, which repeats the code within its subdiagram until a stop condition occurs, executes the case structure and uses the case selector to effect transition to the appropriate case. Each case of the case structure corresponds to a state of the state machine and the case structure's shift registers are used to pass data from one state to the next state.

Further details of the holon implementation are given by [8].

3 INTER-HOLON COMMUNICATION

3.1 Requirements

Holons have the ability to act independently, but also to collaborate to achieve common objectives. Communication between holons is therefore a critical function.

Inter-holon communication is inherently asynchronous since one of the key aspects of a holonic system is that each holon can operate independently, but also collaborate with other holons to achieve a common objective. Each holon should therefore be able to receive a message from one holon and keep it in a message queue, whilst communicating with another holon or performing other tasks. Furthermore, asynchronous communication is necessary so that the sending holon's execution is not blocked while waiting until the receiver is ready to process the message.

LabVIEW automatically divides applications into multiple threads [10], but this is not apparent to the users or programmers. To allow multithreaded operation, all holon interconnections (i.e. the inter-holon communication) has to be compatible with running each holon in its own thread.

3.2 Implementation

Various LabVIEW methods for transferring data between VIs can be considered for inter-holon and intra-holon communication. The method used in each situation depended on the continuity of data, the need for a buffer, the number of variables to be passed and the type of data to be shared [11].

One form of inter-holon communication was already mentioned in Section 2.2: notifiers were used for simple communication actions employing Boolean values such as the occurrence of error conditions and shut-down notifications [8]. The coms holon is, in addition to the external communication discussed below, responsible for stopping the station controller when an error occurs or when the operator hits the *stop* button. To achieve this, *Notifier operations*, found under the Synchronization palette in LabVIEW were used. The coms holon uses the *Send Notification.vi* to send a *true* Boolean value as a notification to other holons. The other holons receive the notification via the *Get Notifier Status.vi*. By wiring the notification terminal of each holon to the conditional terminal of the holon's while-loop, the station can be shut down.

More complex inter-holon communication was implemented using LabVIEW's queues and third-party LabVIEW XML functions for constructing and parsing messages. Asynchronous communication using queues is based on LabVIEW's producer/consumer architecture [9].

With the producer/consumer architecture, one holon can send (produce) messages to other holons (consumers). The queues acted as FIFO message buffers for inter-holon communication (and in some cases for intra-holon communication). LabVIEW provides convenient functions to add a message (here in the form of XML strings) to a queue, to remove the oldest message, to view the oldest message without removing it from the queue and to check whether there are any messages in the queue. LabVIEW queues can trigger a timeout if a message is not available within a set time, which is a useful feature for diagnosability.

LabVIEW queues therefore allow lossless asynchronous communication between holons, where the sending holon adds messages to the queue and the receiving holon removes the messages when it is ready to do so.

Only one queue could have been used as "inbox" for each holon for inter-holon communication. However, multiple queues were implemented (one queue for each sender-receiver pair) since it obviates the need to read the received message in order to tell which holon sent it. Furthermore, using multiple queues simplifies prioritising the messages, allowing a holon to get the information it needs most urgently without having to first read the other, earlier messages in a queue. Finally, using multiple queues aids in debugging and controller reconfiguration by clearly defining in the architecture where messages come from and go to in each queue. In the implementation here, the queue names were composed of the sending and receiving holons' names, e.g. "ProductInfoReq_from_OH_to_PHM".

Since each LabVIEW queue handles only one data type, all the messages passed through the queues were here formulated as strings in XML format. For the sake of commonality, the XML conversion VIs that were selected for the external communication (described in a later section) was also used for the internal communication.

4 INTRA-HOLON COMMUNICATION

Intra-holon communication is required when holons internally have separate processes running in parallel or sequentially, and these processes have to exchange information.

This intra-holon communication was achieved using

- Queues (similar to inter-holon communication),
- Local variables (for static information)
- By passing data through wires, which are flow paths connecting different function nodes on the block diagram, and
- By shift registers, as mentioned in Section 2.2, where information needs to be passed from one state to a subsequent one.

5 EXTERNAL COMMUNICATION

The station controller must also be able to communicate with the cell controller. In the research presented here, XML strings passed over TCP/IP connections were chosen for this type of communication since it is vendor-neutral and allows communication between controllers implemented on various technology platforms (e.g. C# and LabVIEW). The XML format was also chosen since it naturally makes provision for information that can

be used to improve the robustness of the information exchange.

LabVIEW's built-in XML functions could not be used for the messages exchanged with the cell controller, because some of the XML functions cannot generate or parse XML schemas defined by others, while other functions are unsuitable because of complexity in their use. Therefore, the third-party JKI EasyXML palettes were used to convert LabVIEW data to and from XML strings. These functions were used for both external communication (with the cell controller) and inter-holon communication, to keep the XML schemas consistent.

JKI EasyXML is a LabVIEW toolkit that can be used to create, parse, read, and write LabVIEW data to and from XML [12] and, unlike the built-in LabVIEW XML functions, allows the user to control the XML item names. "Easy Generate XML_JKI EasyXML.vi" and "Easy Parse XML_JKI Easy XML.vi." respectively serialise (on the sender's side) and de-serialise (on the receiver's side) LabVIEW data structures, by converting the LabVIEW data names (labels) to/from XML item names and the LabVIEW data values to/from XML item values. In order to convert the output to the desired data type after parsing, the LabVIEW function *variant to data* is used.

The coms holon, one of the staff holons, handles communication between the station controller and the cell controller. The cell controller sees the station controller as a single module and therefore one holon in the station controller should handle all the communication with the cell controller. Using a separate coms holon also shields the cell controller communication from delays when the other holons are busy, and vice versa. The role fulfilled by the coms holon can be expected to be present in all station controllers that are modules in a distributed control system. However, the protocol used for information exchange between the cell controller and the station controller could take various forms and therefore what is presented here in this respect, should be seen as an illustrative example.

The coms holon maintains two FIFO buffers, the *IN FIFO* and the *OUT FIFO*, in respective while-loops, namely a reader and a writer. The *IN FIFO* that holds messages from the cell controller and the *OUT FIFO* that holds messages to be sent to the cell controller, as illustrated by the flow charts in Figure 3.

The buffer of the TCP/IP socket that receives messages from the cell controller (implemented using a standard LabVIEW method) was used as the *IN FIFO*, while the *OUT FIFO* queue was implemented using the queue functions as with other inter-holon communication. The other holons place their messages for the cell controller in the *OUT FIFO* queue.

The cell controller and station controller communicate through TCP/IP with messages formatted as XML strings. Using the *TCP Read* VI, the coms holon first reads four bytes of data from a TCP/IP connection, which gives the length of the message, and then reads the indicated number of bytes. Similarly, the coms holon uses the *TCP Write* VI to write data from the rivet feeder station controller to a TCP/IP connection with the cell controller.

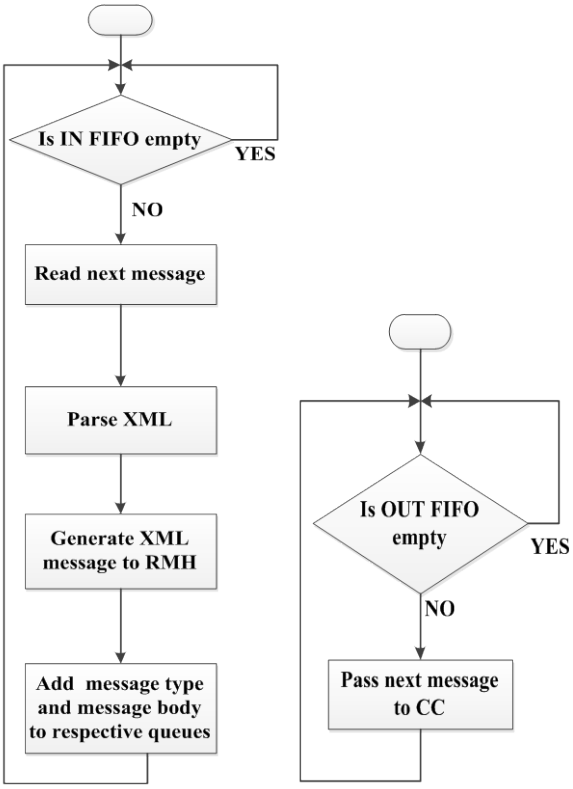

Figure 3 - Coms holon flow-charts

The coms holon is also responsible for stopping the station controller when an error occurs or when the operator hits the *stop* button. To achieve this, *Notifier operations*, found under the Synchronization palette in LabVIEW were used. The coms holon uses the *Send Notification.vi* to send a *true* Boolean value as a notification to other holons. The other holons receive the notification via the *Get Notifier Status.vi*. By wiring the notification terminal of each holon to the conditional terminal of the holon's while-loop, the station can be shut down.

6 COMMUNICATION WITH PHYSICAL DEVICES

LabVIEW's ability to interface easily with a wide range of National Instruments (NI) products, is a distinct advantage in using LabVIEW for a station controller. NI products allow the control program to run on a PC and also on stand-alone controllers, such as the compactRIO-9068 (cRIO-9068) used for the case study here. This 8-slot integrated controller and chassis system provided an interface between

the LabVIEW based controller and the rivet feeder station resources, i.e. the Pick-n-Place mechanism and the XYZ positioning table. The controller used various digital inputs and outputs, operating at the popular 24 V level, to sense proximity sensors and to actuate pneumatic control valves. The case study also used a CANopen module to interface the controller with the Festo servo drives of the XYZ positioning table. Further details of the case study implementation are given in [8].

7 ASSESSMENT

Since the focus of this paper is a station in an RMS cell, an assessment was performed to determine to what extent the key characteristics of RMSs (flexibility, scalability, convertibility, modularity, integrability and diagnosability, as mentioned in Section 1) are demonstrated by the LabVIEW-based controller. The assessment was performed by carrying out experiments using the case study and by considering relevant LabVIEW features. Due to paper length constraints, the details of the assessment are given by [8] and the assessments are only briefly described here.

7.1 Changing the product type

In the first experiment, the impact of a change in product was tested. The experiment showed that introducing a new product, that does not require any new processes, required no changes in the station controller and hardware. The new product type could be accommodated without any station controller changes since the station controller is not limited to pre-programmed product types and the relevant production information was similar to the product family considered when designing the station.

7.2 Adding a new device to the station

Convertibility and scalability, the distinguishing features of RMSs, require the ability to reconfigure a manufacturing system so that new production technologies can be incorporated, new product types can be produced and the system's production capacity can be changed. Convertibility and scalability rely on modularity and integrability, also in the control system.

Therefore, it was important to assess the LabVIEW-based controller's convertibility and scalability, and this was done by adding a new functional element to the rivet placing station. In the controller, the reconfiguration led to the addition of a new resource holon, the singulation unit holon, and to changes to the order holon.

The addition of the new singulation unit holon and the changes to the order holon, including establishing the interfaces with the digital outputs to the solenoid valve, took about three and a half hours to complete.

7.3 Diagnosability

Diagnosability can play various roles, including warning higher level control systems when inconsistent production instructions are received or when a fault has occurred in the station resulting in its unavailability for production.

Faults in the station are diagnosed in the case study by checking for timeouts while waiting for operations (e.g. movement by the XYZ table holon) to be completed. In LabVIEW, timeout checking is easily performed by specifying the time, in milliseconds, that the function waits for an element (information) to become available in a queue. When a timeout occurs, LabVIEW activates a particular execution path and it is relatively simple to direct the holon to an error state, where a message to the cell controller can be generated and the station controller commanded to shut down.

In the third experiment, this ability to generate timeout errors was confirmed by deliberately setting an unreasonably short timeout for one action and confirming that an error message was sent to the cell controller, that an error message was also displayed to the operator and that the station was triggered to stop.

8 CONCLUSION

This paper contributes to the RMS research by evaluating the suitability of LabVIEW's communication functions for developing a controller for a reconfigurable manufacturing station. The paper shows that a holonic control architecture can be implemented to achieve reconfigurability of the rivet feeder station.

LabVIEW's communication functions hold the following advantages:

- LabVIEW's queues provide a convenient way to implement lossless asynchronous complex communication.
- Network-published variables and TCP/IP components simplify communication over a network, allowing easy distribution of parts of the controller.
- Third party methods are available to work with custom XML formats in LabVIEW.
- LabVIEW can easily communicate with hardware, such as digital I/Os, real time controllers (such as the compactRIO) and the main industrial communication protocols (such as the CANopen card used in the case study).

Reconfigurability assessments conducted on the case study showed that the key characteristics of RMSs were demonstrated. We therefore conclude that LabVIEW is suitable for implementing a station's control system in a reconfigurable manufacturing cell.

9 ACKNOWLEDGEMENTS

The authors wish to acknowledge the contributions of CBI Electric: Low Voltage, who provided case study information, and the funding support of the Department of Science and Technology's ICT initiatives (DST/CON 0089/2014).

10 REFERENCES

[1] Hoffman, A.J., Basson, A.H., 2015, IEC 61131-3-based holonic control of a reconfigurable manufacturing subsystem, under review by Int Jnl Computer Integrated Manufacturing.

[2] Hoffman, A.J., 2014, IEC 61131-3-based control of a reconfigurable manufacturing subsystem, MEng (Mechatronics) thesis, Stellenbosch University, Stellenbosch, South Africa.

[3] Halvorsen, H., 2012, Introduction to LabVIEW. Tutorial, Dept of Electrical Eng, Inf Techn and Cybernetics, Telemark University College.

[4] National Instruments, 2003, LabVIEW User manual.

[5] Koren, Y., 2005, Reconfigurable manufacturing and beyond. CIRP 3rd Int Conf Reconfigurable Manufacturing.

[6] Malhotra, V., Raj, T., Arora, A., 2010, Excellent techniques of manufacturing systems: RMS and FMS. Int Jnl of Engineering Science and Technology 2: 137-142.

[7] Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., Peeters, P., 1998, Reference architecture for holonic manufacturing systems: PROSA. Computers in Industry 37: 255-274.

[8] Masendeke, D.M., 2015, Evaluation of LabVIEW based control for a reconfigurable manufacturing subsystem, MEng (Mechanical) thesis, Stellenbosch University, Stellenbosch, South Africa.

[9] Thuyphamxuan, 2013, LabVIEW design patterns: Software design approaches. [Online] Available from: http://gtms1311.wordpress.com/2013/03/09/labview-design-patterns-software-design-approaches/ [Accessed: 4th February 2015].

[10] Anonymous, 2014, NI Tutorial 6424: Differences between multithreading and multitasking for programmers, National Instruments.

[11] Humayun, S., Mehmood, M., Mahmood, F., UlIslam, Q., 2013, A study and comparison of data transfer methods in LabVIEW. World applied sciences journal 28(11):1772-1775.doi:10.5829/idosi.wasj.2013.28.11.1784.

[12] Anonymous, (undated) EasyXML Toolkit for LabVIEW - JKI, Available from: http://sine.ni.com/nips/cds/view/p/lang/en/nid/209021 [Accessed: 6 February 2015].

11 BIOGRAPHY

Darlington Masendeke obtained his MEng Mechanical from Stellenbosch University in 2015. He is a lecturer at the Copperbelt University, Zambia.

Anton Basson obtained his PhD in Aerospace Engineering at Penn State University in 1992. He is a Professor in Mechanical Engineering and co-leader of the Mechatronics, Automation and Design Research Group at University of Stellenbosch, South Africa.