*Send Orders for Reprints to reprints@benthamscience.ae*





# Software Reconfiguration Patterns for Instrumentation Architectures

Dengpan Zhang and Hongli Zhu*

*Henan Polytechnic University, Jiaozuo, China*

**Abstract:** Dynamic reconfiguration in all kinds of applications is absorbing more and more research focus for its demand in automatic measurement system for low-cost products and short time to market. This study proposes a software reconfiguration instrumentation architecture, which is built on component and software data bus models. In the architecture, the measurement system software is viewed as an integration of a set of reusable components, which are modeled with routing workflow for measurement, data process logic for execution of behavioral specifications and a series of communication ports for data exchange with others. The behaviors of the instrumentation system software can be viewed as an integration of components and their interactions and specified in application specification, which is based on routing workflow and independent of component implementation. When the measurement system needs to be reconfigured to adapt to the changing application requirements, the software using the architecture can be easily reconfigured by changing reusable components and their interactions in application specification and reconfiguration can be achieved at the executable code level after the software is implemented. Finally, a reconfigurable measurement platform based on above software architecture for testing engineering signal has been developed and the goal of higher reconfiguration, lower development and maintenance costs are achieved with the proposed architecture.

**Keywords:** Reconfigurable software, Measurement system, Software bus, Component model, Workflow.

## 1. INTRODUCTION

Software Reconfigurable instrumentation patterns have been presented to adapt to the change of market conditions and requirements of customers. Flexibility, reuse and reconfiguration are the main characters that are defined as the ability to repeatedly change and rearrange the components of a system in a cost-effective way [1, 2].

Instrumentation that serves as the important part of industrial applications tends to be updated and changed many times during its lifetime. Object-oriented model has been addressed to describe, analyze, and design the measurement system and the object model of measurement domain are presented to support the reuse of component, but the object model tends to be simple without considering the evolution of component and the coupling interactions of components [3].

With the rapid development of computer and electronic technologies, virtual instruments were brought forth at the beginning of the 1990s and now are widely used in the measurement fields of industrial products. Several commercially available integrated development platforms, such as LABVIEW and VEE, provide users with a graphically programming way to quickly build all kinds of measurement systems. In these platforms, function components are pre-implemented, but reusability and reconfiguration can only be achieved in the design time [4, 5].

Component-based model, design and integration have been the main focus of software engineering for many years and recently are widely used in industrial measurement and automation application to support reusability and reconfiguration [6, 7]. Some architecture such as common object request broker architecture (CORBA), component object model (COM) and Enterprise Java Beans (EJB) have been presented to support implementations of reuse and reconfiguration, but these models depend on middleware servers and are not suitable for measurement systems due to their high system overhead [8-10] .

Actor model based software architecture explicitly specifies the interaction of components and consists of a set of actors that can be pre-implemented to glue components in the design of real-time system [11, 12]. However, the behaviors of the implemented actors are not reconfigurable. A similar actor concept by adding hierarchy and behaviors are used to make the actors more suitable for applications [13]. All these models are presented mainly for reusing specification in the design phase and without an explicit consideration of reconfiguration after implementation.

Reconfigurable software architecture for measurement system presents a control plan language to describe the reconfigurable behaviors of components and their integration and reconfiguration can be achieved to meet the change of application requirements, but components communicate with others through a routing table inside themselves with a direct way, which makes the reconfiguration of interactions of components become complex [14, 15].

The goal of this paper is to enable the reuse of implemented components in industrial measurement applications. In the architecture, the interactions of components are implemented by workflow model which consists of event-based

*Address correspondence to this author at the Henan Polytechnic University, Jiaozuo, China; E-mail: albert_12@126.com

1874-4443/15 

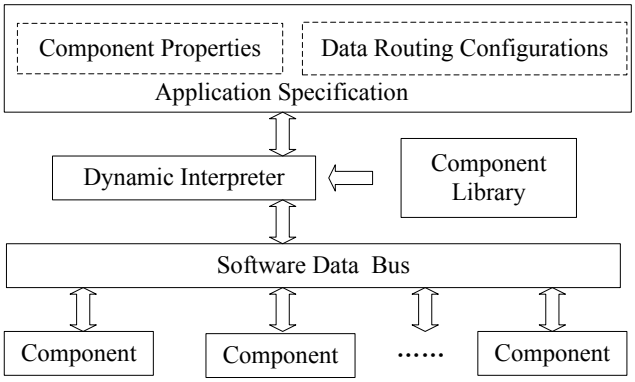



**Fig. (1).** Reconfiguration architecture of the instrumentation.

external interfaces, components can be structurally integrated into the system and communicate with others with the routing table in the software bus.

The main valuable contribution of this paper lies in separating application specifications, including component configurations and interactions of components by workflow from the routing models, so that different applications can be reconfigured independently when the system is in run-time.

The rest of this paper is organized as follows: Section 2 describes the architecture for building the reconfigurable instrumentation system, including hierarchy software data bus structure, component model and application specification with dynamic interpreter mechanism. Section 3 describes the behaviors of reconfiguration and the implementation with the example. Section 4 draws some conclusions and presents the future work.

## 2. THE ARCHITECTURE OF THE INSTRUMENTATION SYSTEM

### 2.1. The Reconfiguration Management Framework

The reconfiguration of the instrumentation allows the measurement platform to be changed while the measurement process is in execution, which can be looked as data stream processing system. The reconfiguration can be abstractly viewed as change to components properties and data flow sequences. The former describes the functionality of components and the latter defines the interactions of components. Reconfigurable instrumentation is a component-based system developed in C/C++, where data driven applications are assembled from components communicating via workflow events. The workflow of the events is considered as application specification, which are addressed in our previous research [15].

The instrumentation system consists of pre-built measurement component library, dynamic script interpreter and software data bus, as shown in Fig. (**1**). In the architecture, application specification is customized with script that provides control over the measurement process.

The script describes measurement sequences at a high-level abstraction and supervises execution by sending control events to components, which processes or visualizes the data. Script can be dynamically interpreted and components are assembled into the platform. RMS (Reconfigurable Measurement System) is expressed as a composition of communicating components and the interactions of components, which are implemented by software bus.

Software data bus provides a communication architecture where data-driven components communicate via events and offer dynamic linking of component upon loading and serve as the mediator in inter-component communication.

The component library consists of a set of general purpose components and is supplemented by application domain specific components, which group measurement components and other modules and are unique to various specific application and hardware. Components have been developed for data acquiring, data analyzing and data displaying.

Such a software framework breaks the dependency between the application behaviors and RMS platform. High-level reconfiguration can be achieved and the framework supports the same application specification running on different RMS platform and different application specifications running on the same platform which needs to be reconfigured only when the component library changes and the application specification needs to be altered when the requirement of measurement process changes.

### 2.2. Component Model

A Component is a unit of composition with provided and required interfaces. Similar to an object, a component may encapsulate state and provides interfaces, define services the component offers to others. Components are distinguished from objects in that they are configuration independent, which is defined by explicitly required interface, interaction independence and conforming to a binary standard. Components are pre-implemented software modules and are used as building blocks to construct the measurement software. A component defines the functionality of a device or subsystem, which can be as simple as a device like a data

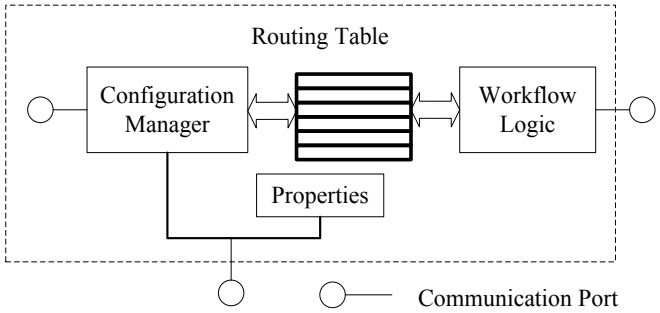



**Fig. (2).** Reconfigurable component model.

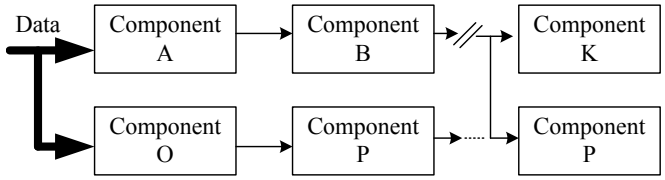



**Fig. (3).** Data exchange patterns.

acquisition board and a signal process algorithm like Fast Fourier Transform Algorithm (FFT) or as complex as composed subsystem like oscillograph.

A component interacts with others to obtain the desired services by routing tables in a dynamic configured measurement system. The services that each component provides are specified as acceptable external events and other components can invoke the desired functions based on component workflows. Such interactions are constructed during design time and can be changed only at some predefined safe state like configuration state during running time. The structure of a measurement software component includes a set of external interfaces with configuration manager and routing tables, communication interfaces with receive and send ports , internal event interfaces with own proprieties management mechanisms and data processing logic of the workflow, as shown in Fig. (**2**).

This model makes the component different from commonly used model such as CORBA, DCOM and EJB, which are usually based on remote procedure calls and heavily depend on predefined middleware services such as naming and look up services.

External interfaces are designed to provide component functions to other components, which define operations that can be invoked from outside and are presented as a set of acceptable events in the workflow model. And they enable operations to be configured and allow components to be intergraded into the measurement system at run-time. A customizable event routing mechanism is devised and added in each component to achieve the relationship with software bus. Such a routing mechanism separates the implementation of component with its interfaces and the routing table can be customized without knowing interactions with other components. A configuration management mechanism is also added to perform run-time monitoring of the type of received events, only those events invoked by authorized and acceptable events will be executed to add or delete the routing relationship with other components. These customizations can be predefined in the application specifications, loaded and modified in run-time.

Communication ports are used to connect components for integration, which serve as physical interfaces of a component. Each communication port has a set of attributes associated with it which define the type of communication port (sending or receiving), data exchange methods (shared memory, event-driven), the way of communication (synchronous or asynchronous), as is shown in Fig. (**3**).

Workflow logic module is the best important part of a component. It receives the data and invokes process functionality immediately and then the result will be packed and transformed to the communication port.

Internal interfaces define channels to access the internal properties of a component and separate internal properties from functionality and support components reconfiguration at executable level. Internal interfaces can be used for users to modify the parameters at run-time. In the architecture, internal interfaces can be divided into displaying and functional interfaces, the former are used to access geometry and colorful parameters and the later are designed to modify function parameters such as data acquired frequency and data acquired channel.

When the system begins to run, each component receives events from its communication ports, external and internal interfaces. The acceptable events are then translated and sent to workflow logic. The workflow logic will perform one or a sequence of function calls and generate result, and then the result will be packed and sent to the corresponding components.

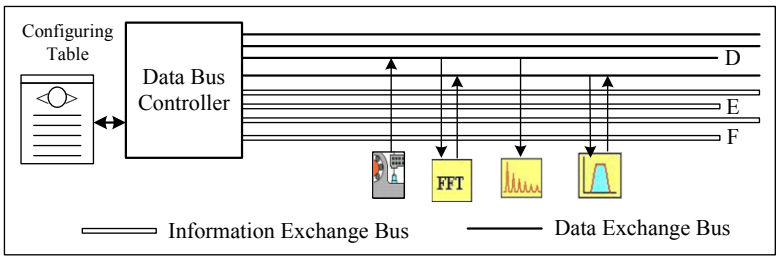



**Fig. (4).** Software data bus model.

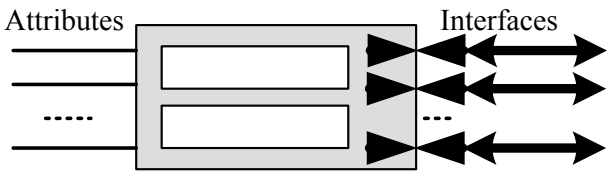



**Fig. (5).** The reconfiguration instrumentation component model.

### 2.3. Software Bus Model

Software data bus serves as the component connector in dynamic reconfiguration application and provides a virtual address routing mechanism for components to communicate with each other. Software bus separates the interaction specifications from component implementation and reconfiguration of interactions of components can be easily achieved at run-time. This makes our connection model different from commonly used model. The structure of software bus includes software bus controller with an interaction routing table, external event interfaces and assembling port.

In a reconfigurable instrumentation, each component needs to configure the interested interactions with the software bus and components are more interested in the types of events. Software data bus controller or management module is designed to manage all response interactions of components in the routing table and it avoids direct connection of components and keeps flexible interactions and facilitates the reconfiguration of instrumentation system.

Assembling port is designed to provide a channel for software data bus controller to load the interaction configuration in routing table (see Fig. **4**). When the analysis of application specification by dynamic interpreter in the design time is finished, the interaction can be dynamically reconfigured with the change of application specification.

External event interfaces are added for the components that are plugged into the software data bus to dynamically change their communication relations with others. The external events consist of the request for adding, modifying and deleting.

In order to configure the instrumentation system flexibly, the data bus is very important. It takes the role of linking components, transferring measurement data and controlling the execution of the program. To implement the data exchange, two extensions are done for the data bus. First, data bus can connect to the components in the current tasks; second, the output data can be sent to all the components which are connected to the data bus, as is shown in Fig. (**4**).

The instrumentation components are designed with special interfaces to connect to the data bus (see Fig. **5**). Attributes are used to define work parameters of the component. Interfaces are used to connect to data bus. In this way, the components can receive data or send data only by interface APIs which is different from the traditional designing patterns.

### 2.4. Dynamic Interpreter Model

Dynamic interpreter is used to extract the information of component properties and interactions relationship step by step from the application specification script. Software reconfiguration platform loads components information according to the globally unique identifier. The components will be uploaded, fixed and registered according to the uniform resource locator included in the script if the identifier information is not found in the operation system. After the loading of components is implemented, the interpreter continues to explain the application specification and components are plugged into the software bus. Fig. (**6**) introduces the dynamic interpreter model of reconfiguration instrumentation platform.

### 2.5. Data Stream Based Control Model

The reconfiguration instrumentation platform includes a series of software instrumentation components, such as data acquiring components, digital filtering components and spectrum components. Each component implements a type of

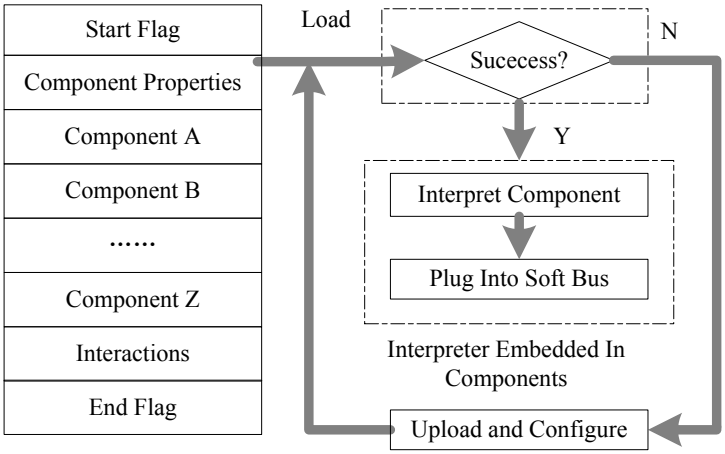



**Fig. (6).** Dynamic interpreter model.

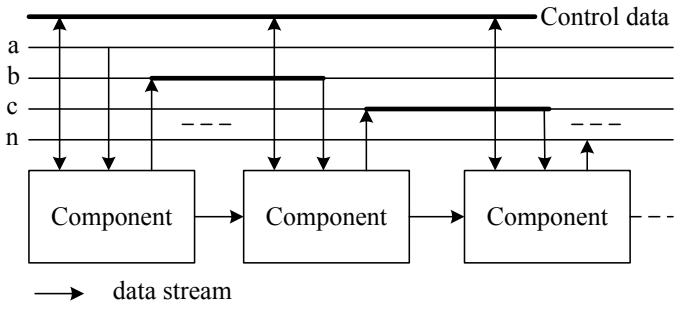



**Fig. (7).** The data stream among the measurement components.

function, such as data processing, data editing and data displaying. These components connect with each other according to the data processing sequences. When the data stream flows through connected components, the data are processed step by step until the end of the components (see Fig. **7**). All the components exchange data based on the control information from the platform and the information is exchanged through control data bus, as is shown in Fig. (**7**).

## 3. DYNAMIC RECONFIGURATION AND IMPLEMENTATION

### 3.1. Dynamic Reconfiguration

When the measurement cases are reconfigured, the platform can produce the text based XML(Extensible Markup Language), which store all the information of instrumentation cases. All the components included in the cases can be classified into four categories, such as indicator components used to display analysis results, control components used to receive user's input, Driver components such as DAQ used to control hardware to get measurement data and Data processing or Algorithm components used to process measurement data.

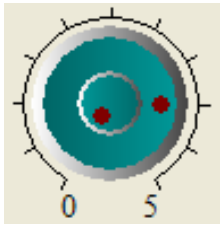
#### *3.1.1. Interface Description of Control Components*

The Knob component is used to receive user's operation information which only writes data to data bus. The language of the component includes attributes that are used to define its size, position, color and interface that are used to send user's operation data. According to Fig. (**8**), the center of the knob is (60,60), the radius of the knob is 30, the background color of the component is green, and the pin color is red. The range of user's input is (0 5). And user can send the operation value to the line ID 3.

When the related measurement objects need to be measured, users can send control instructions by uniform protocols to the measurement component, and then the configuration manager interprets the protocols and configures the working patterns. At the same, the manager constructs a new working process to carry out the measurement tasks.

#### *3.1.2. Interface Description of Indicator Components*

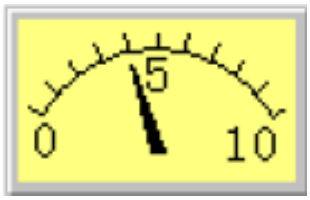
The indicator component is used to display analysis result. It only receives data from data bus. Its XML tag includes attributes that are used to define its size, position and color, and terminals that are used to receive measurement data. For example, Fig. (**9**) is a XML tag of meter. The script




```
<Knob>
<Center value="60 60"/>
<Radius value="30"/>
<BgColor value="Green"/>
<FgColor value="Red"/>
<MinMax value="0 5"/>
<OutLine value="3">

</Knob>
```

**Fig. (8).** Knob component and its reconfiguration tag.




```
<Rectangle Meter>
<Keyword value="meter"/>
<Position value="10 10 110 75"/>
<Bg value="light yellow"/>
<FgColor value="black"/>
<InLine value="3">

</ RectangleMeter >
```

**Fig. (9).** Rectangle meter component and its reconfiguration tag.

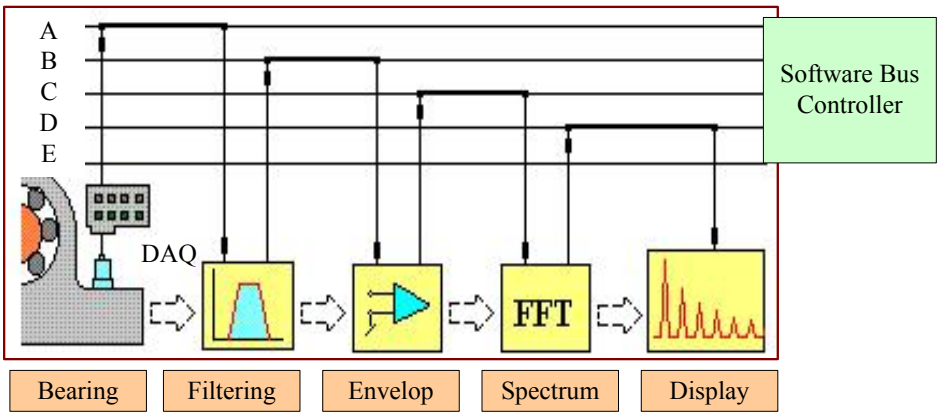



**Fig. (10).** The Reconfigurable measurement and diagnosis system.

assembles a meter in the front panel of the platform. The left, top, right and bottom of the meter are 10,10,110 and 75. The background color of the meter is light yellow. The text color and pin color of the meter are black. The meter receives measurement data from the pipeline 10.

### 3.2. The Process of Dynamic Reconfiguration

Reconfiguration of instrumentation system consists of structural changes, including addition, removal and replacement of components, as well as system reorganization.

Fig. (**10**) is the block diagram of a bearing diagnosis system. The system is divided into the data acquisition component, band-pass filter component, envelop-processing component, power spectrum component and display component. The vibration signal of the bearing are acquired, processed and transmitted along components according to the measurement workflow.

Components additions and removals are necessary when new devices and signal process arithmetic are integrated into the systems. Such reconfiguration can be done by adding or removing the corresponding components. For component additions, the modification of exist system is minimized if the new added components can communicate with existing components. The event routing and register mechanism of new added components may need to be realized to satisfy their interactions with other components. On the other hand, removal of a component requires updating the routing table of acceptable events, which are used to communicate with the removed components.

Components may need to be replaced when an existing device or signal process subsystem in the platform is replaced. Replacement can be achieved with a component removal followed by a component addition and easily realized if the added component has the same configuration as the removed one, otherwise, the new component and the

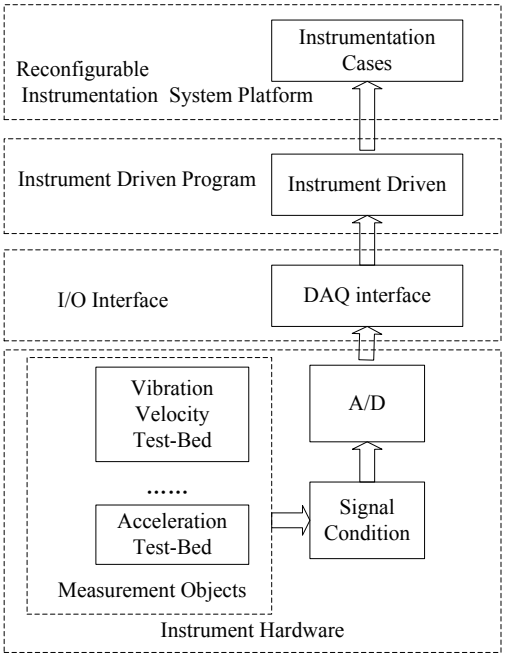



**Fig. (11).** Reconfiguration architecture of instrumentation system.

components it communicates with, as well as the event routing and register information, need to be reconfigured.

System reorganization requires changes of interactions of components. It usually occurs when the relationships between components change or when platform changes. With the software data bus and component model in our framework, reorganization of component relationships can be achieved by modifying the corresponding routing address of software bus and reorganization of platform configurations can be achieved by customizing the application specifications.

### 3.3. The Implementation of Dynamic Reconfiguration

The instrumentation platform has been developed for all kinds of experiences of engineering signal processing and is composed of instrument hardware, I/O interface, instrument driven program and software development environment, as shown in Fig. (**10**).

In the platform, the traditional development process has been replaced by a new one, which uses a component-based technology. Using this technology, systems are assembled from a set of components without traditional programming effort, as shown in Fig. (**11**). The selected components adapt to their roles by setting their properties.

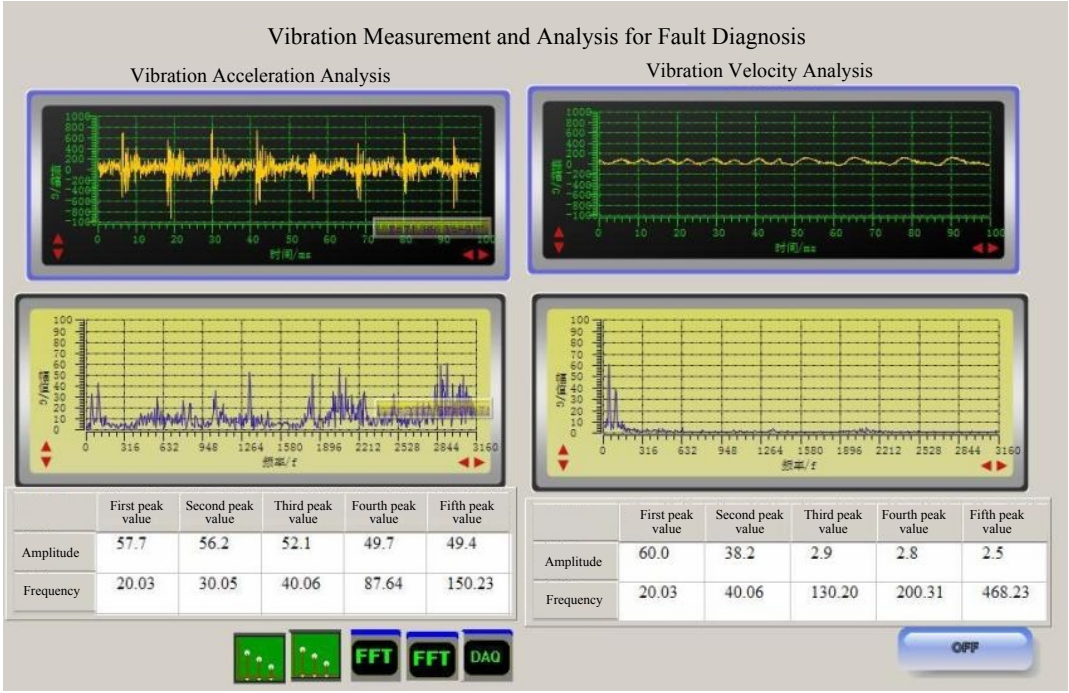
A series of measurement programs have been constructed using the platform. The cases consist of hardware-objected components, data processing components, data visualization components and data archival components.

When the requirements of users are changed, the reconfiguration can be easily achieved by modifying application specification. A measurement system for vibration measurement is easily achieved by defining a new application specification, as is shown in Fig. (**11**).

### 3.4. Application Example

The platform using software reconfiguration patterns in instrumentation design is developed for industrial application. To better highlight the obtainable improvements with the proposed methods for instrumentation application, the implementation of the application is described.

In this section, the reconfiguration case of vibration measurement and analysis for fault diagnosis is shown. In this case, the components include one DAQ(Data Acquisition), two signal processing units, four wave information showing units, and two data listing units, *etc*., (see Fig. **12**).




| | First peak value | Second peak value | Third peak value | Fourth peak value | Fifth peak value |
|---|---|---|---|---|---|
| Amplitude | 57.7 | 56.2 | 52.1 | 49.7 | 49.4 |
| Frequency | 20.03 | 30.05 | 40.06 | 87.64 | 150.23 |

| | First peak value | Second peak value | Third peak value | Fourth peak value | Fifth peak value |
|---|---|---|---|---|---|
| Amplitude | 60.0 | 38.2 | 2.9 | 2.8 | 2.5 |
| Frequency | 20.03 | 40.06 | 130.20 | 200.31 | 468.23 |

**Fig. (12).** Instrumentation cases of reconfiguration patterns.

```
<?xml version="1.0" encoding="UTF-8"?>
<XVICL>
<Controls/>
<ControlButton1 ControlID="70001" X="1023" Y="766" Width="113" Height="43" DataLine="1" ArrayDataLine="0" DataLineOut="0" maxValue="-1" Text="text"/>
<ControlWaveform1 ControlID="70002" X="91" Y="192" Width="500" Height="200" DataLine="-1" ArrayDataLine="5" DataLineOut="0" maxValue="-5000" Text="Amplitude/G"/>
<ControlWaveform2 ControlID="70003" X="92" Y="424" Width="500" Height="200" DataLine="-1" ArrayDataLine="11" DataLineOut="0" maxValue="-1" Text="text"/>
<ControlWaveform1 ControlID="70004" X="624" Y="195" Width="500" Height="200" DataLine="-1" ArrayDataLine="6" DataLineOut="0" maxValue="-5000" Text="Amplitude/G"/>
<ControlWaveform2 ControlID="70005" X="641" Y="427" Width="500" Height="200" DataLine="-1" ArrayDataLine="21" DataLineOut="0" maxValue="-1" Text="text"/>
<ControlList ControlID="70006" X="92" Y="651" Width="500" Height="100" DataLine="-1" ArrayDataLine="12" DataLineOut="0" maxValue="-1" Text="text"/>
<ControlList ControlID="70007" X="648" Y="645" Width="500" Height="100" DataLine="-1" ArrayDataLine="22" DataLineOut="0" maxValue="-1" Text="text"/>
<CControlDAQ ControlID="70008" X="719" Y="773" Width="44" Height="42" DataLine="1" ArrayDataLine="5" ArrayDataLineII="6"DataLineOut="2" maxValue="-1" Text="text"/>
<ControlPeakVal ControlID="70009" X="550" Y="771" Width="44" Height="42" DataLine="11" ArrayDataLine="12" DataLineOut="0" maxValue="-1" Text="text"/>
<ControlSignalFFT ControlID="70010" X="611" Y="772" Width="45" Height="41" DataLine="-1" ArrayDataLine="-1" DataLineOut="0" maxValue="-1" Text="text"/>
<ControlSignalFFT ControlID="70011" X="667" Y="772" Width="43" Height="44" DataLine="5" ArrayDataLine="11" DataLineOut="0" maxValue="-1" Text="text"/>
<ControlPeakVal ControlID="70012" X="491" Y="770" Width="47" Height="42" DataLine="21" ArrayDataLine="22" DataLineOut="0" maxValue="-1" Text="text"/>
<ControlTextH ControlID="70013" X="0" Y="0" Width="0" Height="0" DataLine="-1" ArrayDataLine="0" DataLineOut="0" maxValue="-1" Text="Vibration Measurement and Analysis
for Fault Diagnosis"/>
<ControlTextH ControlID="70014" X="0" Y="0" Width="0" Height="0" DataLine="-1" ArrayDataLine="0" DataLineOut="0" maxValue="-1" Text="Vibration Acceleration Analysis"/>
<ControlTextH ControlID="70015" X="0" Y="0" Width="0" Height="0" DataLine="-1" ArrayDataLine="0" DataLineOut="0" maxValue="-1" Text="Vibration Velocity Analysis"/>
</XVICL>
```

**Fig. (13).** The instrumentation document.

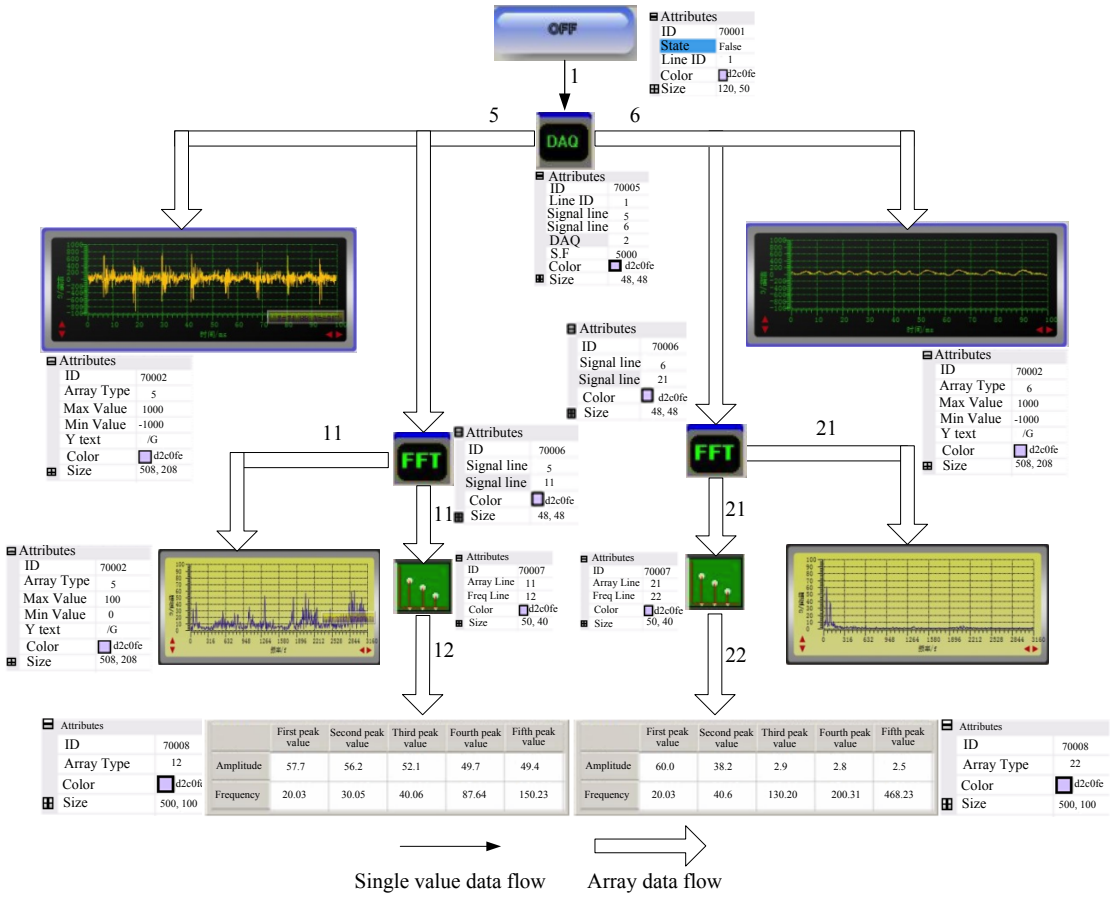



**Fig. (14).** The reconfiguration process of instrumentation case and its data flow.

At first, according to the measurement requirements, the required components in the component library are selected and initialized. Based on the data flow of the measurement process, all the components are linked and the logical linking information is managed by the platform.

In case of the instrumentation application, all the measurement components are configured to work in the auto mode, so at its running stage, all the components can be driven by the measurement data flow automatically. If the data arrives, the data bus managing module of the platform informs the relative components to receive data and process it. As the variety of the input variables is defined, so the interfaces between the linked components are same in the data style. Fig. (**13**) and Fig. (**14**) present the reconfiguration process of the measurement instrumentation.

During the reconfiguration process, the software components required are selected and the attributes of selected components are configured. All the components have input

interfaces and output interfaces, the two components which exchange data must have the same kind of data type. On the other hand, the management module of the platform schedules the data exchange among the components according to the data bus line. When the instrumentation case is built, the platform can finish the text document (XML) of the instrumentation which can be read and interpreted by the platform. Fig. (**13**) is the total text document based on the XML language.

The reconfiguration software instrumentation can acquire the vibration signal from the production device; the signal includes vibration acceleration data and vibration velocity data. Based on the data bus, the value can be shown as wave, and then the data can be processed by the FFT components which can transfer the time-domain data into frequency domain. The characteristic parameters list is given in list components. When the applied requirements of users are changed, the reconfiguration can be easily achieved by modifying application specification.

## 4. CONCLUSION

In this paper, we presented a kind of instrumentation architecture based on software reconfiguration patterns for constructing reconfigurable measurement system. Such instrumentation reconfiguration will enable adaptation to the change of application requirements. In the architecture, reconfigurable software consists of communication components that are modeled with a set of workflow based extern interfaces, internal interfaces, data process logic and a series of communication ports. Behaviors of software reconfiguration platform are specified in application specification routing roles and reconfigurable workflow logic. Hierarchy software data bus and composition component model are addressed to achieve high-level reconfiguration and data flow model is presented to realize data exchange of components. Structure and behavior reconfigurations can be achieved by changing the composition of components and modifying the application specifications respectively. Reconfiguration without structural changes inside the existing components, does not require code regeneration and executable code level reconfiguration can be easily achieved. A series of measurement programs running on the instrumentation platform for testing engineering signal has been developed and can be reconfigured to meet the changing requirements of users, which show that such software is more flexible, reused and reconfigurable.

## CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

## ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation of China (Grant No.50275061) and the National High Technology Research and Development Program (Grant No.2008AA04Z133).

## REFERENCES

[1] M.G. Mehrabi, A. G. Ulsoy, and Y. Koren, "Reconfigurable manufacturing systems: Key to future manuf.," *J. Intell. Manufac.*, vol. 11, no. 4, pp. 403-419, 2000.

[2] P. Spicer, and H. J. Carlo, "Integrating reconfiguration cost into the design of multi-period scalable reconfigurable manufacturing systems," *J. Manuf. Sci. Eng.-Trans. ASME*, vol. 129, no. 1, pp. 202-210, 2007.

[3] Q. P. Yang, and C. Butler, "An object-oriented model of measurement systems," *IEEE Trans. Instrum. Meas.*, vol. 47, no. 1, pp. 104-107, 1998.

[4] S. Kohout, J. Roos, and H. Keller, "Automated operation of a homemade torque magnetometer using LabVIEW," *Meas. Sci. Technol.*, vol. 16, no. 11, pp. 2240-2246, 2005.

[5] D. Hyun, and J. Kim, "Study of external humidification method in proton exchange membrane fuel cell," *J. Power Sources*, vol. 126, no. 1-2, pp. 98-103, 2004.

[6] W. Kozaczynski, and G. Booch, "Component-based software engineering," *IEEE Softw.,* vol. 15, no. 5, pp. 34-36, 1998.

[7] W. M. P. van der Aalst, K. M. van Hee, and R. A. van der Toorn, "Component-based software architectures: a framework based on inheritance of behavior," *Sci. Comput. Program.*, vol. 42, no. 2-3, pp. 129-171, 2002.

[8] I. Gorton, and A. Liu, "Evaluating the performance of EJB components," *IEEE Internet Comput.*, vol. 7, no. 3, pp. 18-23, 2003.

[9] D. N. Gray, J. Hotchkiss, S. LaForge, A. Shalit, and T. Weinberg, "Modern languages and Microsoft's component object model," *Commun. ACM*, vol. 41, no. 5, pp. 55-65, 1998.

[10] T. F. Lunney, and A. J. McCaughey, "Component based distributed systems - CORBA and EJB in context," *Comput. Phys. Commun.*, vol. 127, no. 2-3, pp. 207-214, 2000.

[11] I. Crnkovic, and M. Larsson, "Challenges of component-based development," *J. Syst. Softw.*, vol. 61, no. 3, pp. 201-212, 2002.

[12] D. B. Stewart, R. A. Volpe, and P. K. Khosla, "Design of dynamically reconfigurable real-time software using port-based objects," *IEEE Trans. Softw. Eng.*, vol. 23, no. 12, pp. 759-776, 1997.

[13] M. Moallem, "Design and implementation of computer control software," *IEEE Control Syst. Mag.*, vol. 25, no. 1, pp. 26-29, 2005.

[14] S. G. Wang, and K. G. Shin, "Task construction for model-based design of embedded control software," *IEEE Trans. Softw. Eng.*, vol. 32, no. 4, pp. 254-264, 2006.

[15] L. He, and D. Zhang, "XVIML: an extensible virtual instrument markup language," In: *IEEE AUTOTESTCON 2005*, Orlando, Florida, Sep 26-29, 2005, pp. 36-42.

Received: September 16, 2014 Revised: December 23, 2014 Accepted: December 31, 2014