The Design and Implementation of Digital Signal Processing Virtual Lab Based on Components*

Jianxin Wang[1], Lijuan Liu[1], and Weijia Jia[2]

[1] School of Information Science and Engineering, Central South University, ChangSha, 410083, China
jxwang@mail.csu.edu.cn
[2] Department of Computer Engineering and Information Technology, City University of Hong Kong, Kowloon, HongKong
itjia@cityu.edu.hk

Abstract. This paper proposed the design and implementation of digital signal processing virtual lab (DSPVL) based on components. In the DSPVL, all the virtual instruments are developed as components and implemented as Java Beans, which improve the developing efficiency, the reuse of software and make the system be maintained and expanded easily. This paper also introduces the characters and architecture of user platform and illustrates the key design and implementation technologies. In the DSPVL, we developed a lot of components for the experiments in Digital Signal Processing (DSP) course. In this paper, we also gives an example of designing components of Discrete Fourier Transform (DFT) to show the process of designing, implementing and using the components in the DSPVL.

Keywords: Virtual Laboratory, DSP, components, Java Beans.

1 Introduction

With the rapid development of Internet, modern long-distance education as a new education mode has became an important problem for discussion. Virtual Laboratory (VL) based on the Internet is a key on improving the quality of distance education since experiments are significant for most engineering and application courses [1,3].

Digital Signal Processing (DSP) course is an important course in information science. How to construct a security, powerful and easy-use Digital Signal Processing Virtual Lab (DSPVL) platform is the key part of our research. In order to keep the balance of high efficiency and correctness during developing the DSPVL, we propose a new method that all the virtual instruments are encapsulated in the components based on Java Beans which can keep the platform independent and make users interact with each other well [2]. By choosing the required components to construct an experiment, users can do DSP experiments and deeply understand and consolidate the complex knowledge.

* This work is supported by the Major Research Plan of National Natural Science Foundation of China, Grant No.90304010 and City University Strategic Grant No. 7001587.

R.W.H. Lau et al. (Eds.): ICWL 2005, LNCS 3583, pp. 291 – 301, 2005.

The rest of the paper is organized as follows. In section 2, we introduce the architecture of DSPVL. Section 3 describes the design and implement of DSPVL modules. Section 4 deals with the design and implementation of the components in DSPVL. Conclusion is given in section 5.

2 The Architecture of DSPVL

The DSPVL is designed based on BS mode [2]. Its client is implemented with components and object-oriented technologies that offer users with applet embed in HTML files. The server side mainly includes Web Server to visit DSPVL through browsers. The architecture of the DSPVL is shown in Fig.1.

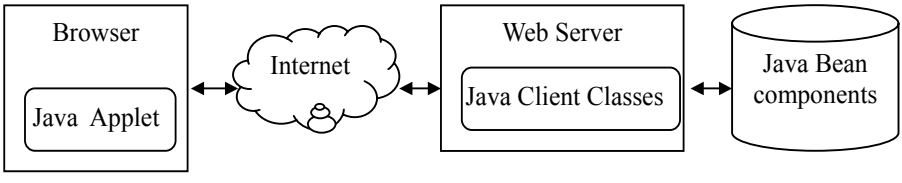

Fig. 1. The architecture of the DSPVL

Its clients contain many components that have two types. The first is to invoke the objects and methods in the server side and run, the second is to run directly in client side. Besides using the components offered in DSPVL to do experiments, users can also submit their self-develop components to the Web Server from the client side. After the system administrator testing the self-develop components strictly, they can be inserted to the system and all the following users can use them. By this way, the DSPVL will be more applicable and scalable.

3 Design and Implementation of DSPVL

The DSPVL is constructed of three modules that are experiment designing, experiment running and submitting components. In design module, users select and connect the virtual instruments to construct an experiment. The default parameters of components are set by the system, if necessary users can reset them. Then the DSPVL will run the experiment and display the result in run module. When users want to use and test their local components developed by themselves, they can add them into the DSPVL in submission module.

3.1 Implementation of Design Module

Fig.2 shows the main modules in the DSPVL. Users connect Web Server through browsers. After entering the DSPVL, browsers automatically download applets from server to clients. There are several classes in the user interface in the DSPVL.

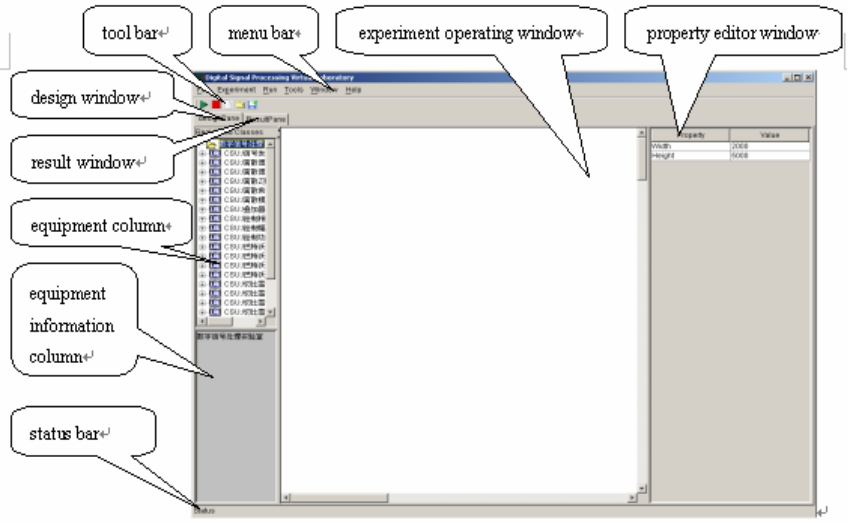

Fig. 2. Main modules in DSPVL

(1) MainWindow Class

This class realized the main frame of the platform. It contains design window, result window, tool bar and menu bar. In menu bar, we set six buttons named file, experiment, run, tools, window and help to do experiment work. All the buttons are defined in MainMenu class by the method of setJMenuBar. In order to trigger and perform the events by pushing the buttons, we set some listeners and actions. The main code is showed as follows:

```
Public class MainWindow extends JFrame{
    mainMenu=new MainMenu(this);
    toolbar=new MainMenu(this);
    designPanel=new DesignPanel(this);
    resultPanel=new ResultPanel(this);
    jTabbedPane=new JTabbedPane();
    jTabbedPane.add(designPanel, "designPane");
    jTabbedPane.add(resultPanel, "ResultPane");
    this.getContentPane().add(StatusBar,BorderLayout.
    SOUTH);
    this.getContentPane().add(jTabbedPane,
    BorderLayout.CENTER);
    this.getContentPane().add(toolBar,
    BorderLayout.NORTH);
    this.setJMenuBar(mainMenu);
    actionPerformed(ActionEvent e);
    }
```

(2) RegisteredClassPanel Class

XML(Extensible Markup Language) is structural, scalable and self-definitional, so that XML can be very useful to create, read, write and save a file. In

RegisterClassPanel class, we define an XML file to register components to the system. By reading this kind of files, we can easily get the parameters and methods of components. The main code of reading components information from XML file is showed as follows:

```
Public void readXML(){
   URL url=null;
   InputStream in=null;
   url=new
   URL((String)csuSystem.getObject("SP_SYSTEMCLASS");
   in=url.openStream();
   Document listXML;
   listXML=XmlDocment.createXmlDocument(in,false);
   NodeList classList;
   classList=listXML.getElementsByTagName("class");
   int classListLength=classList.getLength();
   ClassNode[] classNodes=new
   ClassNode[classListLength];
   for(int loop_class=0;loop_class<classListLength;
   loop_class++)
   {
     Element cur_class;
     cur_class=(Element)classList.item(loop_class);
     DefaultMutableTreeNode cls=null;
     String cls_name=cur_class.getChildNodes().item(1).
              getChildNodes().item(0).getNodeValue();
     String
   cls_title=cur_class.getChildNodes().item(3).
              getChildNodes().item(0).getNodeValue();
     String
   cls_label=cur_class.getChildNodes().item(5).
              getChildNodes().item(0).getNodeValue();
     String cls_desc=cur_class.getChildNodes().item(7).
              getChildNodes().item(0).getNodeValue();
     cls=new DefaultMutableTreeNode(new
       ClassNode(cls_name,cls_title,cls_label,cls_desc,
            classNode.CLASS));
    }
 }
```

(3) PropertyEdit Class

PropertyEdit class is used for users to set the parameters of components and check them whether they are appropriate. If not, the editor will catch the exception and cancel the setting and then ask users to reset them. In addition, PropertyEdit class also supplies methods for users to select that are applied by components.

(4) DesignPanel Class

DesignPanel class is the key class in design module. As a container, it lays an instrument column on the left, operating window in the middle and property editor on the right in the platform. The definition of DesignPanel class is as follows:

```
public class DesignPanel extends JPanel implements
   MouseListener, MouseMotionListener, KeyListener
```

It realizes the interfaces of MouseListener, MouseMotionListener and KeyListener. So by listening to the interfaces, we can make components communicate with each other synchronously. In DesignPanel class we also definite an inner class named DrawCanvas to invoke the interface to link the selected components and realize an experiment flow.

hotLeadArea is a property of DesignPanel class that stands for the component legs of mouse current location. Modifer is the legs' property that records the type of transmission data. By comparing the Modifers between input and output components, we can judge whether they can be connected. If they are the same type, we can invoke method of repaint from an object of DrawCanvas class to connect the legs of components. All the connecting lines are put into the object named connectors inheriting from HashMap class. The main code of realization is showed as follows:

```
set entries=carriers.entrySet();
    Iterator iterator=entries.iterator();
    DeviceCarrier car1;
    Lead h1;
    Map.Entry entry;
    while(iterator.hasNext()){
       entry=(Map.Entry)iterator.next();
       car1=(DeviceCarrier)entry.getValue();
       h1=carl.getLead(e.getPoint());
       if(h1!=null){
          Point[] pa=tConnector.getHandler();
          String  inModifer=
                this.selectedCarrier.getLead(new
                Point(pa[0].x,pa[0].y-3)).modifer;}
          if(!h1.modifer.equals(inModifer)){ break;}
     }
```

(5) DeviceCarrier Class and DeviceConnector Class

DeviceCarrier class stands for equipments and Deviceconnector class stands for connecting lines between components. The most important property of DeviceCarrier class is instance. When the equipments are selected and dragged into the operating window, the system will use the instance to build an object of Java Bean and get its information, and then use static method in Introspector class to return an object defined by BeanInfo to save the parameters and methods of components. At last we can build the legs of components based on the information in BeanInfo. This is the application of reflection technology in Java. By using the technology, object is introduced, loaded and created dynamically. The main code of self analyze for Java Bean is showed as follows:

```
if(newClassURL.toUpperCase().startswith("CSU:/")){
  className=newClassURL.substring(5);
  selfClass=Class.forName(className);
  instance=selfClass.newInstance();    }
BeanInfo
  beanInfo=Introspector.getBeanInfo(selfClass,stopClass);
PropertyDescriptor
  properties[]=beanInfo.getPropertyDescriptors();
Method getter,setter;
for(int i=0;i<properties.length;i++){
  setter=properties[i].getWriteMethod();
  if(setter!=null){
     inCount++;
     l=new lead();
     l.type=Lead.PROPERTY_IN;
     l.offsetX=inCount*10;
     l.name=properties[i].getName();
     l.modifer=properties[i].getPropertyType().getName();
     v.add(1); }
  }
```

3.2 Implementation of Run Module

Run module is the key part for the DSPVL to simulate the process of an experiment. Users can freely choose the required virtual instruments and link them to assemble an experiment flow. Output components can be connected with multiple input components as long as their interfaces are matched. System will build a directed chart with no ring based on the simulate process. The chart uses components as its nodes and connecting lines as its sides [4]. Then analyze the data type between the components and get its topology sequence. If the components can run concurrently, the system will build a single thread for computation. At the same time the system will actuate a management thread to make them work together and respond to users' interrupt instruction. The simulation process is shown in Fig.3.

The key part of run module is ResultPanel class. Its method of queueFlow is used to make the components to queue in a line. The main code is as follows:

```
private void queueFlow(){
  notReayCarriers=new Vector();
  notReadyConnectors=new Vector();
  runQueue=new Vector();
  initNotReadySet();
  while(true){
     String carName=getZeroInCarrier();
     if(carName!=null){
        runQueue.add(carName);
        notReadyCarriers.remove(carName);
        removeConnector(carName);
      }
     else {return;}
    }
  }
```

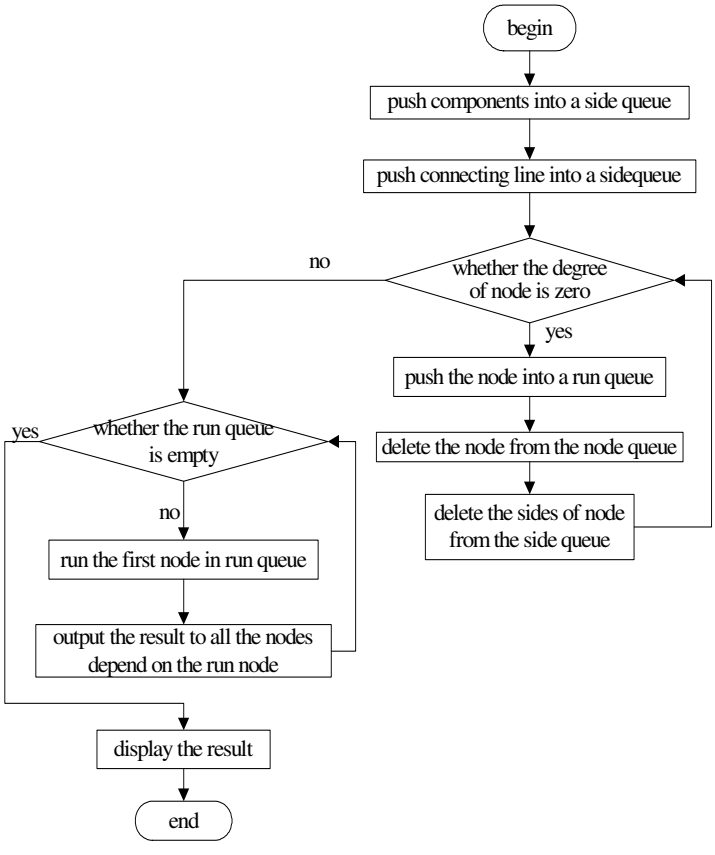

Fig. 3. Flowchart of the simulation process

3.3 Implementation of Submitting Components

This module is used to submit components developed by users. The components must be developed according to the regulations of Java Beans and their interfaces should be matched. Then the users can add their new components into the DSPVL and do experiments with other existent components together.

The process of submission is showed in Fig.4. URLClassLoader class loads the Java Beans to the system and Introspector class gets their information and returns an object defined by BeanInfo class. The object saves the parameters and methods of the components and sent them to another object defined by RegisteredClassPanel class. Then the object will register the new components into the component column. This is the process of how to add the self-developed components by users to the DSPVL. Users can do experiments to test and evaluate them. In order to enhance the function of DSPVL, we use another method to submit components to Web Server by browsers. The administrator checks the correctness and security of components and decides which will be registered. All the qualified components will be added into the system and be offered to all the users.

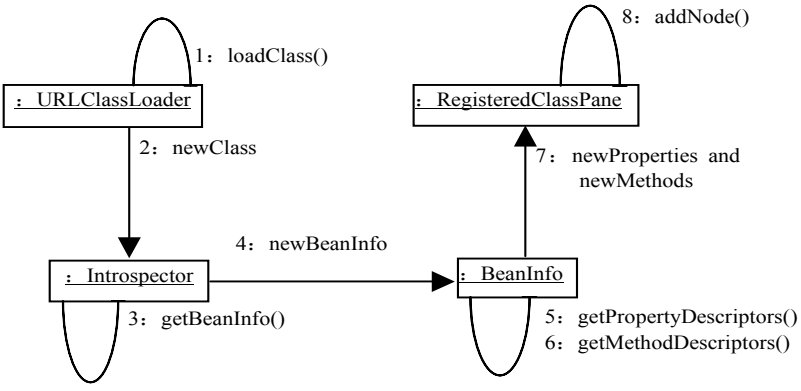

Fig. 4. Submitting components

4 Design and Implementation of DSPVL Components

4.1 General Principle of Designing DSPVL Components

In the DSPVL, we develop many components such as discrete signal generator, signal adder, discrete random signal generator, oscillograph, DSP, discrete Z transform, discrete hilbert transform, amplitude spectrum, angle spectrum, power spectrum and various filters of finite impulse response and infinite impulse response [5]. All the signals are discrete in DSP experiments, so we set the type of transform date among components as a double array. The simulation process is shown in Fig.5. Furthermore, we definite an attribution named sleepInterval to control the run frequency of threads and display the result dynamically.

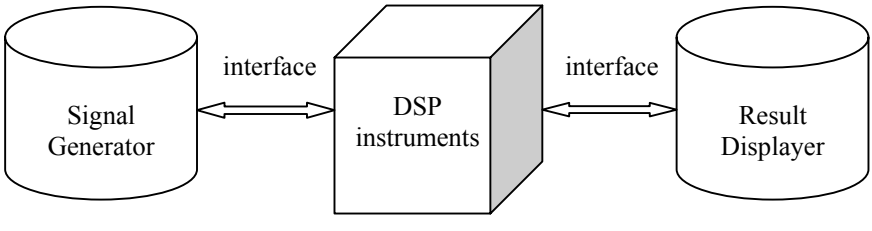

Fig. 5. Simulation process of DSP experiments

4.2 Example for Designing Components

There are many components in the DSPVL. Take the DFT experiment for example to introduce the design and implementation of components.

The algorithm of DFT is Fast Fourier Transform (FFT). FFT algorithm first rearranges the input signals in bit-reversed order and then builds the output to transform based on base-2 select in time domain. The basic idea is to break up a transform of length *N* into two transforms of length *N/2*.

The signal generator generates 32 discrete signals to keep the number of signals is integrated power of 2 and saves them in a double array. There are 5 Java Beans shown in Table.1 that is used in the DFT experiment.

Table 1. The Java Beans Classes in DFT

Class name	Function
sp_Complex	define a type of complex and construct methods of getting amplitude and angle.
sp_DiscreteFourier	realize FFT algorithm
sp_DFT	realize DFT algorithm
sp_SpectrumAmplitude	draw the amplitude of transformed signals
sp_SpectrumAngle	draw the angle of transformed signals

The 32 discrete signals will be sent to the component of sp_DFT from the signal generator through the interface of a double array. The size of array is 33, which use the last position as a flag to judge whether a group of signals has been transferred into sp_DFT completely. Then in sp_DFT class, we define an object named DFT from sp_DiscreteFourier class to invoke the FFT methods. First we reverse the order of the signals. The result is saved in an array named data1 whose type is double.

```
data1 = DFT.reverse(value);
```

Then we invoke the method in sp_DFT class to transform the signals saved in data1 based on base-2 select in time domain and save the result in an array named data3. The type of data3 is complex which defined by sp_Complex class.

```
data3=DFT. transform (data1, data2);
```

The IDFT is the reverse process of DFT and both are based on FFT algorithm, so we construct an all-purpose Java Bean named sp_DiscreteFourier to realize the two transforms respectively in time domain and in frequency domain. In frequency domain, we will reverse the order of signals whose type is complex. So in the method of transform there are two formal parameters, which data1 stands for the real part and data2 stands for the virtual part. In order to make the method of transform be all-purpose, there are also two formal parameters in DFT, but the value of data2 is zero.

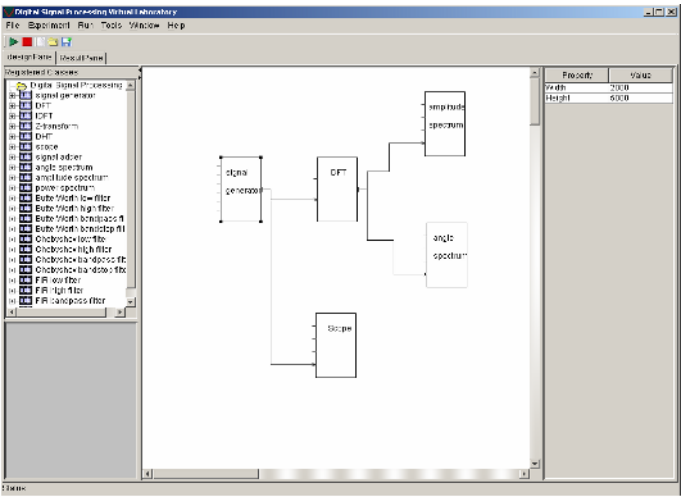

Fig. 6. The simulation process of DFT

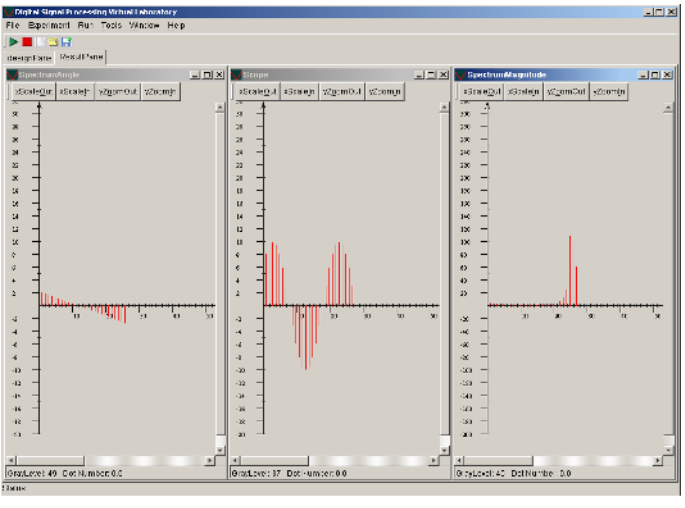

Fig. 7. The result of DFT

Now we get the result of DFT, which the type of data is a complex saved in an array named data3. In order to analyze the result, we send the result into sp_SpectrumAmplitude and sp_SpectrumAngle components by the methods offered in sp_Complex class to get their amplitudes and angles. At last, we'll draw them and see the dynamical result in result window.

```
buffer[i] = data3[i].mod();
buffer[i] = data3[i].arg();
```

The experiment flow is showed in Fig.6 and the result is showed in Fig.7. In Fig.7, from left to right, they are the pictures of angle spectrum of transformed signal, original discrete signal of sine wave and amplitude spectrum of transformed signal.

5 Conclusions

This paper introduces the design and implementation of DSPVL based on components in detail. All the components are developed by pure Java language, which make the system be maintained and extended easily. With the reuse of software and general structure of executing orderly, we can construct various kinds of virtual laboratories (VLs) such as image processing, digital communication and computer network. We still have a lot of work to do to consummate the system and make it more powerful. The VLs will play an important role in the development of remote education, especially for engineering and application courses.

References

1. Wang Jianxin, Peng Bei, Jia Weijia, Design and Implementation of Virtual Computer Network Lab Based on NS2 in the Internet, Proceeding of ICWL 2004, Lecture Notes in Computer Science 3143, 2004, 346-353
2. Wang Jianxin, Chen Songqiao, Jia Weijia, Pei Huiming, The Design and Implementation of Virtual Laboratory Platform in Internet, Proceedings of ICWL 2002 , Hong Kong, 2002.8, 160-168.
3. Jiannong Cao, Alvin Chan, Weidong Cao, and Cassidy Yeung, Virtual Programming Lab for Online Distance Learning, LNCS 2436, First International Conference, ICWL 2002 Hong Kong, China, 2002, P. 216-227.
4. Wang Jianxin, Lu Weini, Jia Weijia, A Web-Based Environment for Virtual Laboratory with CORBA Technology, International Journal of Computer Processing of Oriental Languages, 2003, 16(4):261-274.
5. Steven W. Smith, Ph.D. The Scientist and Engineer's Guide to Digital Signal Processing, California Technical Publishing, 1997, P. 141-168.