ISSN 1009 – 3095 *Journal of Zhejiang University*(*SCIENCE*) V.2, No.4, P.411 – 415, Oct. – Dec., 2001
http://www.periodicals.com.cn; http://www.zju.edu.cn;
http://lib.zju.edu.cn/English; http://www.zjupress.com

VIRTUAL INSTRUMENT SYSTEM SOFTWARE ARCHITECTURE DESCRIPTION LANGUAGE*

ZHOU Hong(周　泓), WANG Le-yu(汪乐宇)

(*Department of Instrumentation Science & Engineering, Zhejiang University, Hangzhou 310027, China*)

Received June 30, 2000; revision accepted Apr.18, 2001

Abstract: In software engineering, an architecture description language (ADL) is intended to aid designers in defining software architectures in terms of abstractions that they find useful, and in making a smooth transition to code. Based on ADL, the concept and models of the Virtual instrument system Software architecture Description Language (VSDL) is provided in this paper. The VSDL put forward provides a new method for virtual instrument system's application design and development by describing the virtual instrument system software architecture effectively. In this paper, the model description、component description and line description are analyzed in detail, and the structure language based on the model is also provided. VSDL provides a smooth interface to graphic software platform, and has been applied to many virtual instrument systems' integration and already yielded good results both in technology and economy.

Key words: architecture description language (ADL), virtual instrument, software architecture description language, graphic software platform, system integration

Document code: A　　**CLC number:** TP311

INTRODUCTION

In order to describe the software architecture of a system correctly, improve the efficiency of system software design and development, software engineers brought forward many methods for analysing software architecture and design models (Garlan et al., 1992; Le Metayer, 1998; Ying, 2000). One of the most intuitionistic methods adopts boxes-lines diagram to develop abstract description. The box represents system software component and the line represents the connection between the components. The method of this model is described as Eq.(1).

$$S = \{B_1, B_2, \cdots, B_n, L_{B_1B_2}, L_{B_1B_2}, \cdots, L_{B_{n-1}B_n}\} \quad (1)$$

This method is comparatively intuitionistic, but is not clear enough in describing either component or connection. To the component, it only gives the abstract concept description without implementation manner. To the connection, it does not give specific definition. What on earth represent does the connection is not clearly defined. It can represent data stream, function stream, property inherit, property include and belongs to process call. Normally, within a system, many connection manners are included and this makes the system software architecture very blurred. Therefore, boxes-lines diagram is only suitable for primary software analysis work.

With the development of Object-Orient technology, which has become the popular method of software design and development, software components appear in concept as encapsulation of class and object, the whole software system is made up of the integration of many classes and objects (Booch, 1986; Coad, 1992; Terry, 1994); communication between the objects is achieved by member functions, which are defined in the class, as shown in Eq.(2).

$$S = \{O_1, O_2, \cdots, O_n, I_1, I_2, \cdots, I_n\} \quad (2)$$

By the cognizance of the object in the software analysis phase, this method confirms the class and the construction of the class layer in the problem space. In the design phase, through definition of the class and organization of class layer architecture, the method confirms the class

* Project(97-772-01-06) supported by "Ninth-Five" National Important Science & Technology Research Project.

and class layer construction; and if they exist in space, confirms the external interface and main data structure. Compared with abstract diagram definition, it is more widely applied; all data structures and operation of class definition can be implemented in concrete programming language. But in the concrete virtual instrument system, it is not easy to cognize the class and the class layer construction, especially when instrument engineers are requested to abstract the problem space to the realizable class (Waheed et al., 1998; Fu et al., 2000). The workload is enormous. What is more is that because data interface is encapsulated in the classes, the communication between data is indirect. This to some extent, leads to difficulties in the software design and program.

In order to describe the software architecture model better, software engineers put forward a new concept "Software Architecture Description Language" using construct text to describe models and connection in software architecture. For example, Rapide based on event model (Luckham et al., 1995), ArTek based on structure design has already been implemented successfully in certain systems (Allen et al., 1994).

PRESENTATION OF VSDL

Virtual instrument system is a typical computer system. It is composed of several modules, following a certain destined goal of information management and implementing specified information test, analysis process and control task. Virtual instrument system is composed of system hardware structure and software structure. Various interface technologies provide connection and interaction between them. System hardware structure has more clear specification, but the design of software structure can embody more capability and flexibility of the whole system; whose scale and quality depends mainly on software structure.

This article presents a new language "Virtual instrument system Software architecture Description Language (VSDL) to describe more effectively the virtual instrument system's software structure based on analysis of the system's software structure features synthetically (Wiederhold, 1992; Kerth, 1995; Allen et al., 1994; Prieto-Diaz et al., 1986).

VSDL is composed of Software Structure Components and Interface-Module, as shown in the Eq. (3).

$$S = \{C_1, C_2, \cdots, C_n, I_{C_1C_2}, I_{C_1C_3}, \cdots, I_{C_{n-1}C_n}\} \quad (3)$$

VSDL is based on the following theoretical foundation:

The system software structure should have the following features for processing software structure description: detachability, independence between software structure components, independence in operation of components. The virtual instrument system software structure uses function layers between upper and lower layers, and left and right layers and adopts a module scheme. So the whole system can be completely separated. Data operation proceeds via the interfaces of software components which have the uniform format.

The virtual instrument system's task is definitely defined and the structure is clear. System integration engineers can decompose tasks and functionality easily. The decomposing can result in description language without class encapsulation, so the design process is comparatively easy.

Virtual instrument developers, during hardware modules design always use hardware description language such as VHDL (Berge et al., 1992). VSDL and VHDL are similar in structure, so it is easy for instrument developers to learn and master them.

Graphics platform, which is used to develop system application program, adopts text file format in internal description. The resulting text, which adopted VSDL to describe software structure is very similar to Graphics Platform's software structure description text. The description result text of VSDL can be used as source code by the graphics platform after parse.

VSDL itself is not a compilable language. Its software description text is more abstract than the description text of the graphics platform. Executable codes are not included in VSDL and executable parts are not included in components. It is parsing and execution proceed via the definition of components class. Thus, VSDL only provides interface between virtual instrument software structure and graphics platform, and its imple-

mentation eventually depends on the special graphics platform.

COMPOSITION AND CONSTRUCTION OF VSDL

VSDL is composed of module description, component description and connection description. Component description and connection description are necessary, while module description is the extended part which is optional. The whole language structure adopts structure text. Program text is composed of a lot of paragraphs, with each paragraph having a start symbol at the beginning and stop symbol at the end, and includes a lot of attribute items. Paragraphs are of three types: module paragraph, component paragraph and connection paragraph. In the module paragraph, basic attribute paragraph, input port paragraph, output port paragraph, and components set paragraph are defined. In the component paragraph, basic attribute paragraph, input port paragraph, output port paragraph and self-defined attribute paragraph are defined. In the connection paragraph, the connection relation between components is defined. The attribute item is composed of left and right item separated by the symbol of " = ". the left item is the name of attribute and the right is the value of attribute. The whole construction description language is shown in follow Fig.1.

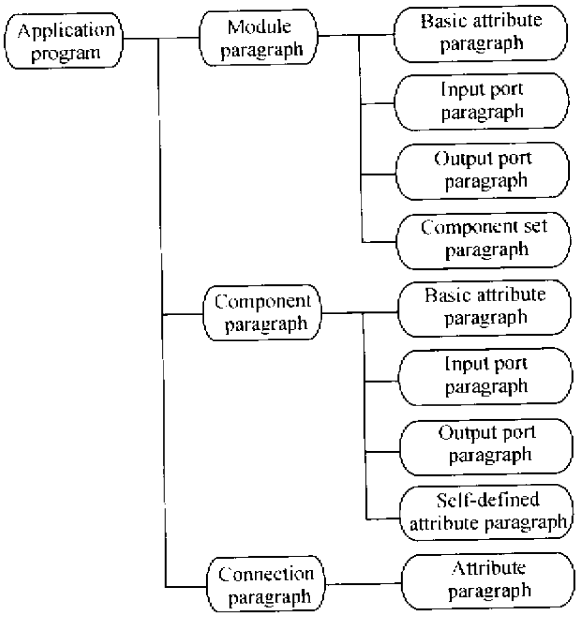

Fig.1 Construction diagram of VSDL description language

Module paragraph uses string "MODULE {" as paragraph start symbol, and "MOUDLE}" as end symbol, and includes a series of attribution paragraph:

Basic attribute paragraph: The paragraph uses "BASEATTR {" as paragraph start symbol, and "BASEATTR}" as paragraph stop symbol. The main items include the module identifier which to identify diverse modules and the module display name used in the flow chart display.

Input port paragraph: The paragraph uses "INPORT {" as paragraph start symbol, and "INPORT}" as end symbol. The main items include the input port name, the data structure, the connection status and the connection name.

Output port paragraph: The paragraph uses "OUTPORT {" as paragraph start symbol, and "OUTPORT}" as end symbol. The main items include the output port name, the data structure, the connection status and the connection name.

Component set paragraph: The paragraph uses "INCOMSET {" as paragraph start symbol, and "INCOMSET}" as end symbol. The main items include all components' identifier in the module.

Component paragraph uses character string "COMPONENT {" as paragraph start symbol, and "COMPONENT}" as paragraph end symbol, in which is a series of attribution paragraph.

Basic attribute paragraph: The paragraph uses "BASEATTR {" as paragraph start symbol, and "BASEATTR}" as paragraph end symbol. The main items include component identifier, component display name, component type identifier, component display size and display position.

Input port paragraph: The paragraph uses "INPORT {" as paragraph start symbol, and "INPORT}" as end symbol. The main items include the input port name, the data structure, the connection status and the connection name.

Output port paragraph: The paragraph uses "OUTPORT {" as paragraph start symbol, and "OUTPORT}" as end symbol. The main items include the output port name, the data structure, the connection status and the connection name.

Self-defined attribute paragraph: The paragraph uses "SELFATTR {" as paragraph start symbol, and "SELFATTR }" as end symbol.

The main items include defined related specified attributes of components, such as limited value of thermometer.

Connection paragraph uses "LINE {" as paragraph start symbol, and "LINE }" as end symbol, in which is a series of attribution paragraph:

LINENAME: Connection name

LINEFROMUNIT: Connection start module or component

LINETOUNIT: Connection terminate module or component

LINEFROMPORT: Connection start port

LINETOPORT: Connection terminate port

Application of VSDL

When using VSDL for program description, the top-down method is adopted. The module paragraph is first defined, then the module unit is decomposed to yield detailed component paragraph definition. The module paragraph can be nested. Connection paragraph aims at both module and component. Ultimately the program is implemented by component types' identifiers which are defined in the component paragraph. The construction of the whole program is shown in below:

```
MODULE {
  BASEATTR {
    NAME = MODULE1
  DISPLAYNAME =
  BASEATTR}
    INPORT {
    PORTNAME =
    PORTTYPE =
    PORTSTAT =
    LINENAME =
  INPORT}
  OUTPORT {
    PORTNAME =
    PORTTYPE =
    PORTSTAT =
    LINENAME =
  OUTPORT}
  INCOMSET {
    NAME1 = COMNAME1
    NAME2 = COMNAME2
      .
      .
      .
  INCOMSET}
  MODULE}
  .
  .
  .
COMPONENT {
  BASEATTR {
    NAME = COMNAME1
    DISPLAYNAME =
    TYPE = VI-TEXT
    LEFT =
    TOP =
    WIDTH =
    HEIGHT =  BASEATTR}
  INPORT {
PORTNAME =
    PORTTYPE =
    PORTSTAT =
    LINENAME =
    INPORT}
  OUTPORT {
  PORTNAME =
    PORTTYPE =
    PORTSTAT =
    LINENAME =
  OUTPORT}
  SELFATTR {
      .
      .
      .
    SELFATTR}
  COMPONENT}
.
.
.
LINE {
LINENAME =
  LINEFROMUNIT =
  LINETOUNIT =
  LINEFROMPORT =
  LINETOPORT =
  LINE}
.
.
.
```

After system software architecture description is put in VSDL description format, the description text parser developed in this project is used to process the decompose step to VSDL text. The parse arithmetic separates paragraphs from VSDL text, assigning texts which belong to different paragraph to different parse subprogram. After

text is separated, the description text can be conveniently used by software developers to choose corresponding graphics platform environment. Now HP VEE developed by HP in USA and VPP developed by Digital Technology and Instrument Institute in Zhejiang University are recommended. Parser encapsulates parse result with VEE or VPP description text automatically. The module description will be converted into UserObject unit in graphics platform. The description text can be directly called by the graphics platform and be converted into graphical source code, with the results stored in graphic platform.

CONCLUSIONS

Use VSDL to describe virtual instrument system software construction yields excellent results in structure and description, and easy interaction with graphics software platform. But unfortunately, if the application program is developed under text language platform, VSDL only provides a system description definition, and can't yield compatible result for language platform. This is VSDL's limitation. VSDL can now only be used to describe the virtual instrument system standard model, not the extend model, and system scale is limited too. At the same time, it is also influenced by defined component types. In some cases, the text described can't be completely convert into source code for graphics platform. So it needs further modification(on the aspects of position and size of components, etc).

VSDL as a new system software construction description method, provides a new way to enhance the efficiency of virtual system instrument software architecture, especially in the design and development of application program. VSDL's practicability resulted in its steady implementation in several virtual instrument system integration projects.

References

Allen, R., Gralan, D., 1994. Formalizing architectural connection. *In*: Proceedings of the Sixteenth International Conference on Software Engineering, p.340 – 348.

Berge, J. M., Fonkoua, A., Maginot, S. et al., 1992. VHDL Designer's Reference. Kluwer Academic Publishers.

Booch, G., 1986. Object-oriented development. *IEEE Trans. on Software Engineering*, SE – **12**(2): 211 – 221.

Coad, P., 1992. Object-oriented patterns. *Communications of the ACM*. **35**(9): 153 – 159.

Fu Lieyong, Gu Zhongwen, 2000. The Development of simupro series operator training systems for complicated industrial processes, *Journal of Zhejiang University* (*SCIENCE*), **1**(4): 377 – 380.

Garlan, D., Kaiser, G. E., Notkin, D., 1992. Using tool abstraction to compose systems. *IEEE Computer*, **25**(6): 30 – 38.

Kerth, N. L., 1995. Caterpillar's fate: A pattern language for the transformation from analysis to design. *In*: James Coplein and Douglas Schmidt, eds., Pattern Languages of Program Design, Addison-Wesley, p.293 – 320.

Le Metayer, D., 1998. Describing software architecture styles using graph grammars. *IEEE Trans. On Software Engineering*, **24**(7): 521 – 533.

Luckham, D. C., Augustin, L. M., John, J. K. et al., 1995. Specification and analysis of system architecture using Rapide. *IEEE Trans. on Software Engineering, Special Issue on Software Architecture*, **21**(4): 336 – 355.

Prieto-Diaz, R., Neighbors, J. M., 1986. Module interconnection language. *Journal of System and Software*, **6**(4): 307 – 334.

Terry, A., 1994. Overview of Teknowledge's domain-specific software architecture program. *ACM SIGSOFT Software Engineering Notes*, **19**(4): 68 – 76.

Waheed, A., Rover, D. T., Hollingsworth, J. K., 1998. Modeling and evaluating design alternatives for an Online instrumentation system: A case study. *IEEE Trans. on Software Engineering*, **24**(6): 451 – 470.

Wiederhold, G., 1992. Mediators in the architecture of future information systems. *IEEE Computer*, **25**(3): 38 – 48.

Ying Jing, He Zhijun, Wu Minghui, 2000. Evolution-based software developing environment, *Journal of Zhejiang University* (*SCIENCE*), **1**(4): 381 – 383.