Tools Cooperation in an Integration Environment by Message-passing Mechanism

Chi-Ming Chung, Ying-Hong Wang, Wei-Chuan Lin, Ying-Feng Kuo

Department of Information Engineering
TamKang University
Taipei, Taiwan, R.O.C.

Gwo-Ching Hsieh

Project Manager
CASE Framework Project
Institute For Information Industry
Taipei, Taiwan, R.O.C.

Abstract

Many CASE(Computer Aided Software Engineering) tools had been developed to increase the software productivity. Therefore, the tool integration needed to be more investigation. This paper proposes an architecture based on a control integration platform for exchanging messages among different tools. This platform supports different tools to integrate into a cooperative software developing environment easily. Some tools had been developed including compiler, editor, testing path analyzer are integrated into this environment. Some evaluation criteria are illustrated to assess the proposed platform and their integrated tools.

keywords: Tool Integraton, Evaluation Criteria, Platform

1 Introduction

Software Engineering is to investigate software development process to promote the software productivity and reduce the developing cost. According to the waterfall model mentioned in [7], the software developing is divided into many phases as depicted in Fig. 1. These phases are analysis of software requirement, definitions of software specification, principles of system design, structured programming techniques (or coding), system and software testing, software product maintenance. Among these phases, many CASE(Computer Aided Software Engineering) tools were developed to provide the automatic or semi-automatic software development methodologies. However, these tools were selected according to user's subjective thoughts from many different tool vendors to form a CASE environment. As these selected tools may be from different tool vendors, the information exchanging among these tools is a big problem. This is because the information needed among these tools may have different format, so the tool integration for providing each developing phase is more and more important.

In this paper, Section two describes the approaches to solve the tool integration problem. Section three explains the message-passing architecture which is composed of ***message-server***, the ***message protocol*** and the ***integration interface***.

Section four introduces the tools which helped user to generate the testing path of developing software automatically. The assessment and advantage of tools integrated into this environment had also been discussed. Section five is the conclusion.

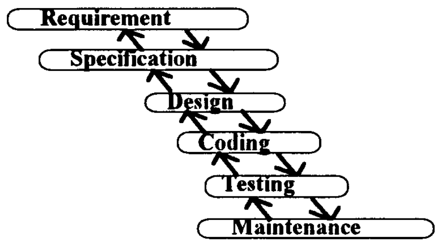

Fig. 1 The WaterFall Model

2 Tool Integration Approaches

Several approaches managed the tool integration are introduced. The first approach is to define a common data format discussed among tools to exchange each other's information. There are three drawbacks in this approach. First, it is difficult to define a common data format which is acceptable to all tools. Second, it is necessary for all tools to modify their source codes to fit this data format or develop the conversion module to convert the data formats among tools[8]. Thus, if there is a new tool which is intended to join to this integrated environment, almost every tool vendor must modify their source code to identify this tool as depicted in Fig. 2. The third is that tools which were used in a developing environment are just execution codes. Because the tools used by users may be developed by different tool vendors, the tool integration is more difficult without the corresponding source codes.

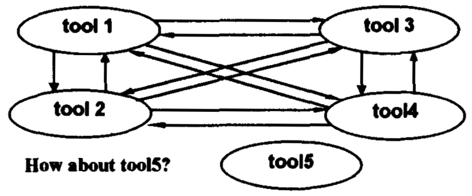

Fig. 2 The first approach of tool integration

The other approach to solve tool integration is accomplished by data integration and control integration[1]. Data integration is to integrate tools through a common database of tool information called common data repository. However, how to let a new tool join in this environment and not to change all of the existed common shared data structure among tools. Control integration is to communicate among tools via message-passing rather than shared common data structure defined in data integration. This message passed among tools is defined as an agreed protocol. Whenever a tool needs the service which is provided by other tool, this tool just sends a request message to a ***message-server***. When the message was decoded, the ***message-server*** resends to the appropriate tool, and does the required service. Therefore, when this action is done, the tool sends the results to the requested tool by packaging them into a message and sending to the ***message-server***. The ***message-server*** decodes the message again and resends back to the requested tool. This process is depicted in Fig. 3.

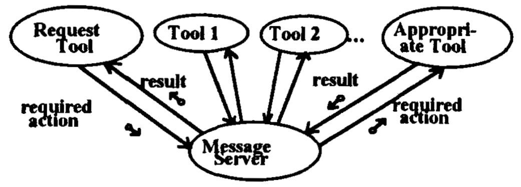

Fig. 3 The other approach of tool integration through message-passing mechanism

The first method proposed to support the control integration is developed in Brown university -- the FIELD environment[15,16]. This environment is based on UNIX operating system and the UNIX processes communicate with each other via socket. The message of the FIELD environment is the first format defined to be sent among different processes. Products developed by HP[2,3] and SUN[2,10,11] have similar message transmission protocols. They all do some enhancements to provide more flexible capability for tools to exchange messages. These environments were also implemented in UNIX operating system which had already provided the abilities of IPC(Inter-Process Communication).

As the Microsoft's WINDOWS system is widely used in the personal computer, this paper proposed an architecture to integrate tools in a Microsoft's WINDOWS environment.. This architecture is based on the idea of control integration and divided into two parts. One is the ***message-server***, the other is the ***message protocol*** and the ***integration interface*** to encapsulate the tools in this environment.

3 The Message-passing Architecture

The UNIX operating system had provided the facilities to support the IPC mechanisms. They are shared memory, semaphore and message-passing. And almost all UNIX operating system had the network ability embedded which are TCP/IP, RPC(Remote Procedure Call), NFS(Network File System) etc. These make the tools distributed in different machine easy to communicate. Besides, the X Window system had been developed in the UNIX operating system for many years. This can make tool easy to be implemented in a common look-and-feel, user-friendly presentation. But when tool vendor had the notice about the importance to cooperate with each other, these developers have to study the application interfaces of IPC, network and X Window supported by UNIX operating system to make their tools integrate together. If these application interfaces had been assembled into an easier environment, this will make tool easier and quicker to be integrated. The same implemented strategies of the products are HP's SoftBench[2,3] and SUN's ToolTalk[2,10,11].

III(Institute for Information Industry) noticed that there are many PCs in the world and the Microsoft's WINDOWS system is the suitable software for programmer to develop a common look-and-feel presentation tool. However, the same problems arise -- the developers had to study the IPC facilities provided by Microsoft's WINDOWS system to communicate among tools. The Microsoft's WINDOWS system provided the DDEML(Dynamic Data Exchange Management Library) to do the jobs of "Inter-Process Communication". This library is composed of 28 application interfaces, 16 transaction types and the job for developers to understand all of these application interfaces and transaction types is difficult[14]. So III had launched a project to develop a tool integration environment in the Microsoft's WINDOWS system and assemble the DDEML into seven easy, understandable and manageable integration interfaces. This can reduce the effort of integrating tools in the Microsoft's WINDOWS system.

These integration interfaces and message format developed by III are followed the proposal[4,12] drafted by COSE(Common Open Software Environment). The COSE is founded by merging CASE Communique and

CASE Interoperability Alliance which are founded by HP, IBM, Informix, CDC and SUN, Digital, SiliconGraphics companies respectively. So the integration interfaces and message format developed in III in the Microsoft's WINDOWS system may communicate with the message format developed in UNIX operating system. All of this process of integration is passing the message from one tool to the ***message server*** in different operating system and exchanging information among different ***message servers*** to other tool to achieve the job of communication. Therefore, the important portion of the control integration is not only the message format and the integration interfaces but also the message server. The ***message server*** developed in III which is called ***CID***(Control Integration Daemon) is based on the facilities provided by Microsoft's WINDOWS's DDE Server to do the job of Inter-Process Communication. And the message format is listed and explained as followed[6]:

(1) Tool_Class : Unique tool name which is registered in the ***CID*** (to identify the tool name to which the suitable message to be sent).

(2) Operation : Tell the ***CID*** about the required operation provided by other tool.

(3) Message_Type : Tell the ***CID*** that this message is requested the service provided by other tool, or waited the event happened in other tool, or sent the request message to the suitable tool and waited for the service result of other tool, then this requested tool can continue its procedure. So there are three types of Message_Type -- **Request, Notice, Request_reply**.

(4) Context_Host : Tell the ***CID*** about the host where the required tool should process the message.

(5) Context_Directory : Tell the ***CID*** about the host and directory where the required tool should process the message.

(6) Context_File : Tell the ***CID*** about the host, directory and file where the required tool should process the message.

(7) Arguments : Tell the ***CID*** about the arguments needed by the service tool to execute its function.

According to the message format described above, the corresponding integration interfaces developed in III is called ***CII***(Control Integration Interface) can also be listed as followed[13]:

(1) CII_ToolClass : Giving a unique tool name for a tool to register in the ***CID*** when this tool initially joins into this environment. For the further message passing, ***CID*** can identify the suitable message to send to this tool. This interface was mapped to the field of Tool_Class of the above message format.

(2) CII_StartRegistration : Register the tool name defined in CII_ToolClass to the ***CID***.

(3) CII_Provide : Tell the ***CID*** about the services provided in this tool.

(4) CII_Listen : Tell the ***CID*** about the interesting events of this tool. For example, the editor can provide basic editing functions of cut, pasting, load file, save file and the version control tool can provide check-in or check-out of this file. The version control tool can also get interested in the event of the save file happened in the editor. When this event happened, the version control tool can be triggered by ***CID*** and automatically done the check-out service of this file. So every tool in this integrated environment, not only can register the services it provided but also the events related to this tool.

(5) CII_MessageObject : Create the Message Object to put the message item of message format discussed above and later send this Message Object to the ***CID*** to trigger the suitable service provided by other tool.

(6) CII_Send : Send the Message Object to the ***CID***.

(7) CII_StopRegistration : When the tool is not necessary to be integrated in this environment, it can use this integration interface to terminate the relationship with ***CID***.

Through the message format and integration interface described above, there are two levels of tool integration in this environment. One is **Loosely-Integration** the other is **Tightly-Integration**. The difference of these two kinds of integration is when a tool needs to integrate into this environment, the later has to modify the source code of this tool but the former does not. No matter Loosely or Tightly integration for a tool to integrate into this environment, the integration interfaces discussed above are the necessary interface to add in a tool. The process of **loosely-integration** is to create a new file source and add the integration interfaces discussed above which are needed for integration. Then this new source uses the concept of ***fork*** in the UNIX operating system to execute this tool. The process of **tightly-integration** is to add the integration interfaces needed for integration in front of the source code of the tool and reproduce the executable code to join into this integrated environment. The processes of the two-level integration are similar to use the integration interface as an envelope to re-package the tool. So these processes can be called as ***encapsulation***. The architecture of ***CID*** & ***CII*** based on the Microsoft DOS & WINDOWS system of PC is depicted in Fig. 4. The ***CID*** is based on the DDE server and the ***CII*** is to re-package the DDEML which the Microsoft WINDOWS system provided.

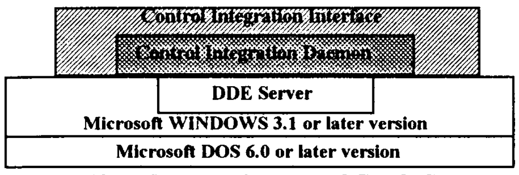
Control Integration Interface
Control Integration Daemon
DDE Server
Microsoft WINDOWS 3.1 or later version
Microsoft DOS 6.0 or later version

Fig. 4 The architecture of CII & CID

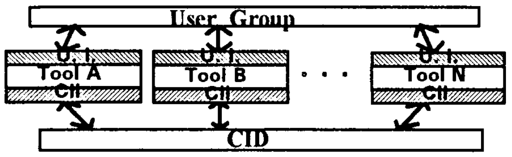

Fig. 5 The final result of tool integration

The run time snapshot of processes is drawn in Fig. 5. In Fig. 5, every tool must be registered in the *CID* to announce that this tool had been joined in this integration environment. This joining procedure is called registration. There are two kinds of tool registration in this environment. One is **static registration** the other is **dynamic registration**. The **static registration** is that tools were added in a system default configuration file(**SDCF**). The entries of SDCF forms the basic tool integration sets in this environment. When the *CID* was invoked, the SDCF is parsed and the basic tool sets is registered in the *CID*. These tool sets are maintained by the *CID* which created an Internal Management Process Table(**IMPT**). After registering into the IMPT, the tool can be triggered automatically by *CID*. This is because when a message was passed to the *CID* and decoded that the receiving tool of this message was not in running state. In this way, the basic tool sets of this environment can be properly managed. The only way for tools in static registration to terminate the relationship with *CID* is to delete the entry in **SDCF** where these tools located. After that, delete the *CID* process and restart it again to reparse the modified **SDCF** to create the new **IMPT**. The **dynamic registration** is to register the tool into *CID* in run time by the integration interfaces -- **CII_ToolClass** and **CII_StartRegistration**. In this way, the tool can be added into the IMPT dynamically. The only way for tools in dynamic registration to terminate the relationship with *CID* is to use the integration interface -- **CII_StopRegistration**. From the above description, it is obvious to see the difference between the static and dynamic registration. The former is to register tool in **SDCF** and terminate the integration of this tool by deleting the entry from **SDCF**. The later added or terminated the integration relationship between a tool and *CID* by the integration interface. The advantage of dynamic registration is that tools can be easily plug-in or replace in this environment. Whereas, the tool added in static registration can be triggered automatically by *CID* when it was not in running state. Fig. 6 depicted the integrated architecture which is proposed by ECMA PCTE to provide an environment for tools to plug-in or replace easily. And the integration environment developed by III is followed this ideal reference model[9].

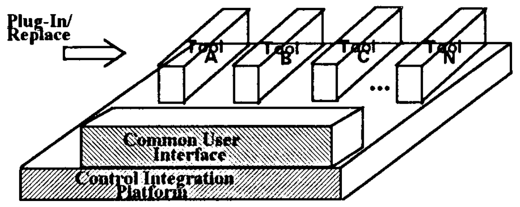

Fig. 6 A control integration architecture referenced the model of ECMA PCTE

4 Evaluation Criteria and Assessment

When *CID & CII* had been designed, the consideration after implementation is to have some basic tool sets to exhibit the advantage of this environment. Therefore, III had developed a coding convention tool and a corresponding output listing tool called CONV and EDITFILE respectively. In order to monitor the status of tool invocation and the flow of message-passing, III also had implemented **CIP(Control Integration Platform)** Manager and Message Monitor tool. The **CIP** Manager parsed the **IMPT** and displayed the entry format of **SDCF** discussed in previous section in the output window. The functionality of Message Monitor is the same, that is to display all of the messages passing back and forth to the *CID*. However, the lack of tools integrated in this environment is still a problem. In the meantime, TamKang university had implemented some testing path analysis tools. These tools originally provided the editing, checking and analysis facilities to software developer. So III and TamKang university had launched a cooperative project to integrate the tools implemented in TamKang university and other companies to provide a convenient environment for users to develop their software. If these tools were not integrated together, the advantage of individual tool is limited. But if they are integrated together, the advantages of these cooperative tools can provide the services which are not existed in each other.

The tools of the cooperative project to be integrated in the integration environment of III are **Editors**, **Testing Path Analyzer(TPA)**, the Microsoft's Visual C++ compiler and tracer(**MSVC & MSVT**), Borland's C++ compiler and tracer(**BLC & BLT**). The tools architecture integrated in the control integration environment developea by III can be drawn as Fig. 7. This figure described the tools integrated in this environment and the corresponding message passing forward and backward. The Editor and **TPA** which were developed in TamKang university are tightly integrated into ***CID*** by modifying the source codes and adding the integration interface in front of them. The **TPA** can be divided as All-Statement Analyzer(**ASA**), All-Branch Analyzer(**ABA**) and Ter. 3 Analyzer(**TER3**). And the **MSVC**, **MSVT**, **BLC** and **BLT** are loosely integrated into ***CID*** by producing new file sources and encapsulating these binary codes into ***CID*** in integration interfaces. When the Editor passed a message containing the program name(**PN**), testing criteria(**TC**), selected compiler(**SC**) or tracer(**ST**) to the ***CID***, the ***CID*** decoded this message and passed to one of three test path analyzers, two compilers or tracers. After completing their service, these tools sent the result or program listing(**PLL**) to the ***CID*** and the ***CID*** resent back to the Editor to point out the corresponding program location(**PL**) of the source code.

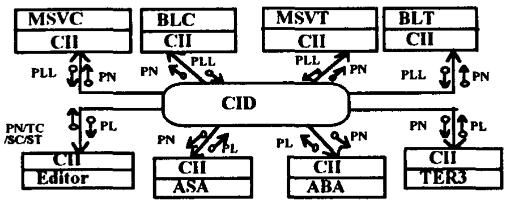

Fig. 7 Tool integration architecture in the platform developed by III

This proposed integration environment can be assessed by an CASE tool criterion. This criterion are compared the HP's Teamwork, III's Kanga tools and this proposed environment from three different aspects: **Individual aspect, Integration aspect, Transformation aspect.**[5] The **Individual aspect** of assessment is to survey if it had provided enough functionality to accomplish the user's tasks. As this environment is the foundation which provided the platform for every tool to be encapsulated, the Individual aspect of assessment can be satisfied by integrating the tools which are necessary to user. The **Integration aspect** of assessment is to select the individual tool which is satisfied to the requirement of user and integrated them together. As tools are integrated into this cooperative environment by the integration interfaces, this environment can fit the Integration aspect of assessment. There are five components to be compared among the HP's Teamwork, III's Kanga tools and this proposed environment. These components are **Common User Interface(CUI)**, **Intertools Data Transfer(IDT)**, **Integrated CASE Repository(ICR)**, **Intermachine Data Transfer(IDT)** and the **Availability On Workstation(AOW)**. The **CUI** is to evaluate that whether every tool integrated in these environment had a common look-and-feel interface or not. The **IDT** is to evaluate that whether every tool integrated in these environment can exchange information among tools or not. The **ICR** is to evaluate that whether these environment possessed the Integrated CASE Repository to store the information of CASE tools or not. The **IDT** is to evaluate that whether the information of CASE tools can exchange with other CASE tools which are in different machine or not. The **AOW** is to evaluate that whether these environment which are supported on workstation or not. And the corresponding result of comparison listed in Table 1. The **Transformation aspect** of assessment is to evaluate the properness of the information exchanging among tools. No matter what kind of tool to be integrated into this environment, the message-passing is controlled by the ***CID***. The ***CID*** can dispatch messages forward and backward properly according to the message type. So this environment can also meet the Transformation aspect of assessment. There are three components to be compared among the HP's Teamwork, III's Kanga tools and this proposed environment. These components are **Interphase Automatic Information Transformation(IAIT)**, **Selected Diagramming Transformation(SDT)**, **Full Diagramming and Information Transferred(FDIT)**. The **IAIT** is to evaluate that whether the information of different phase of tool can be transformed automatically or not. The **SDT** is to evaluate that whether the diagramming tool in these environment can transform a selected diagram into another diagram or not. The **FDIT** is to evaluate that whether the diagramming tool in these environment can transform a selected diagram and its information into another diagram and corresponding information or not. Finally, this compared result listed in Table 2.

Table 1. The Integration aspect of assessment

Integration aspect of assessment	HP's Team-work	III's Kanga tools	Proposed environ-ment
CUI	Support	Support	Support
IDT	Support	Support	Support
ICR	Support	None	None
IDT	Support	None	Will-be
AOW	Support	None	Will-be

Table 2. The Transformation aspect of assessment

Transform-ation aspect of assessment	HP's Team-work	III's Kanga tools	Proposed environ-ment
IAIT	Support	Support	Support
SDT	Support	None	Depending on diagram tool
FDIT	Support	None	Depending on diagram tool

Although this environment is much better than HP' Teamwork and III's Kanga tools, there is still a drawback in this platform. The drawback is when a programmer developing a software system, one thing for him to remember is to cooperate with other tools. That is, once a tool was developed, it is necessary for this tool to open the functions and their corresponding parameters which it provided. In this way, it would be no secret to keep in the individual tool. But this problem can be solved by defining the standard providing functions for tools before they were developed.

5 Conclusion

The architecture discussed in this paper provides a tools integration environment in Microsoft WINDOWS System and makes tools exchange information via message-passing. In this way, a tool does not have to provide all the functions needed for users and can cooperate together to compensate for the drawbacks among tools. This environment also provide the methodology for tool integration, not only limited to the CASE tools but also the other fields of tools just to meet the users' requirements. Therefore, the time required to develop a new tool or modify an existed tool for tool integration can be shorten and with minimum effort.

Reference

1. Alan W. Brown, "Control Integration through message-passing in a software development environment", Software Engineering Journal, May 1993, PP. 121-131.
2. Astrid Julienne, Larry Russell and Brain Fromme, "The Message Is the Medium", SUNExpert Magazine, March 1993, PP 48 - 69.
3. Cagan, M.R., "The HP SoftBench environment: an architecture for a new generation of software tools", Hewlett-Packard Journal, June 1990, PP. 36 - 47.
4. CASE Interoperability Alliance and CASE Communique,"Proposed Messaging Architecture -- A Joint Proposal", December 1992, PP. 1 - 12.
5. Chi-Ming Chung, Po-Yu Chou, "Evalution Criteria of CASE", Journal of Information and Management Sciences, July-August 1993, pp. 51-52
6. Donald Firesmith, "An expanded View of Messages",Journal of Object-Oriented Programming, July-August 1993, pp. 51-52
7. Edward Yourdon, "Modern Structured Analysis", Prentice-Hall International Editions, 1987, pp. 82 - 83.
8. EIA,"CDIF: organization and procedure manual", Report Number EIA/PN-239, January 1990.
9. European Computer Manufacturers Association, "A reference model for computer-assisted software engineering environments", ECMA Report Number TR/55, V2, December 1991.
10. Frankel, R.,"Introuction to the ToolTalk Service", Sun Microsystems Inc., Mountain View, California,1991.
11. Frankel, R.,"The ToolTalk Service", Sun Microsystems Inc.,Mountain View, California, 1991.
12. Fischer, H.,"Notes fromCASE Communique Meeting", 17 October 1991.
13. Hal Dean, "Object-Oriented design using message flow decomposition", Journal of Object-Oriented Programming, May 1991, pp. 21-31.
14. Jeffrey D. Clark, "WINOWS Programmer's Guide to OLE/DDE", Prentice Hall, 1990, PP. 4 - 239.
15. Reiss, S.P.,"Interacting with the FIELD environment", Software Practice & Experience, 1990, 20(S1), PP. 189 - 195.
16. Reiss, S.P., "Connecting tools using message passing in the FIELD environment", IEEE Software, June 1990, PP. 57 - 99.