Program Analysis Environment for Writing COBOL Aspects

Hideaki Shinomi
Advanced Information Systems Research & Development Department
Software Division
Hitachi, Ltd.
5030 Totsuka-cho, Totsuka-ku, Yokohama-shi
Kanagawa 244-8555 Japan
hideaki.shinomi.rh@hitachi.com
and
Hitachi Consulting, Co., Ltd.
2-16-4 Konan, Minato-ku
Tokyo 108-0075 Japan

Yasuhisa Ichimori
Hitachi Systems Value Ltd.,
6-21-12 Minamioi, Shinagawa-ku
Tokyo 140-0013 Japan

ABSTRACT

COBOL is still an important language for building mission critical enterprise systems, and there is huge amount of existing COBOL programs. We have been developing an aspect-oriented COBOL and its development environment. We are applying aspect orientation to strengthening internal control in enterprise information systems. Understanding existing COBOL programs is critical for applying aspect orientation to them, because programmers are not able to write aspects without understanding the existing programs. We have developed an environment for understanding existing COBOL programs and developing COBOL aspects. This paper describes the environment and a experience of applying it to improve internal control implemented in a small information system.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*code inspections and walk-throughs*; D.2.6 [**Software Engineering**]: Programming Environments—*programmer workbench*

General Terms

Languages, Design

Keywords

Aspect-oriented programming, COBOL, program analysis, development environment

1. INTRODUCTION

"Cobol: Not Dead Yet" is the title of a 2006 Computerworld article [12]. The article reports the following Computerworld survey findings:

- To the question "What programming languages do you use in your organization? Choose all that apply", 62% of 352 IT managers replied "COBOL".
- To the question "If your organization uses COBOL, how much internally developed business application software is written in COBOL?", 43% of the IT managers said "more than 60%".

The survey shows that COBOL is still very important for IT departments in enterprises, and there is huge amount of existing programs written in COBOL.

Aspect-oriented programming [10] has been mainly applied to the Java language [18][17]. We have been designing and developing an aspect weaver and its development environment for COBOL programs. The aspect-description language ALCOB and its weaver Vega were described at the seventh International Conference on Aspect-Oriented Software Development (AOSD.08) [13].

We are mainly applying the aspect-oriented technology to *internal control* in accounting and auditing (see [5] for the definitions of internal control). Internal control [6] requires enterprises to collect and analyze evidence logs (e.g., access logs, transaction logs), and to realize higher security of their systems. Currently, internal control in companies is required by laws in many countries. The Japanese counterpart to the Sarbanes-Oxley Act of 2002 [19] came into effect in fiscal 2008. Enterprises are required to modify their systems in COBOL to support the evidence logs. There is many existing COBOL programs used in mission-critical enterprise systems. Programmers are not able to write aspects for internal control without understanding base programs written in COBOL.

This paper describes a tool for analyzing COBOL programs and supporting programmers to write COBOL aspects. The tool is called Altair. Altair supports the programmers to understand COBOL programs by analyzing data flow and control structure of programs, and also supports them to write COBOL aspects semi-automatically. The users are able to develop aspects without detailed knowledge of the syntax of ALCOB. We used Altair to analyze an account information system and to strengthen internal control of the system by writing aspects. This paper reports the trial.

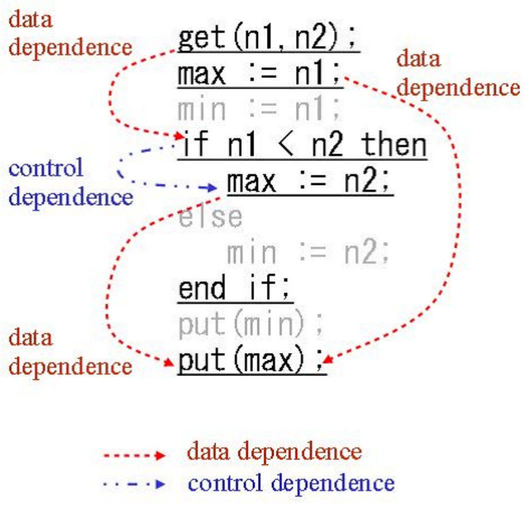


Figure 1: Program slicing example

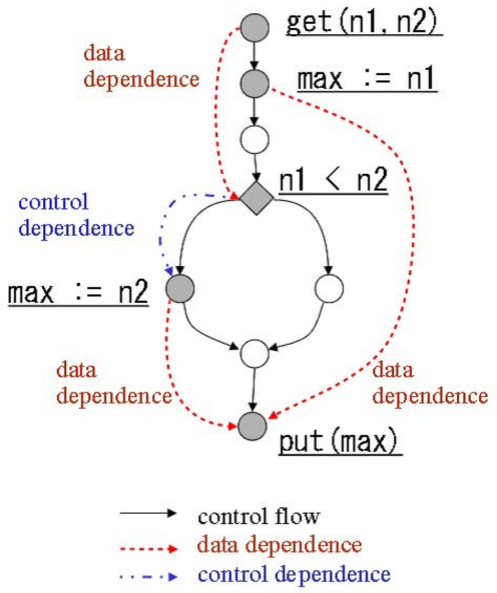


Figure 2: Data and control dependence for the slicing

The rest of this paper is organized as follows. In Section 2, we discuss program slicing technology and its effectiveness for program understanding required to write aspects for internal control. Section 3 describes Altair, the program understanding and development environment for our aspect-oriented COBOL ALCOB. Section 4 presents an experience of applying Altair to write aspects from a standpoint of strengthening internal control. In Section 5, we present the related work. Section 6 gives the conclusion and some future directions.

2. PROGRAM SLICING AND ASPECTS FOR INTERNAL CONTROL

2.1 Program Slicing

Program slicing was first introduced by Mark Weiser [20]. The computed program slice consists of the parts of a program that might affect or be affected by the values computed at some point of interest. The computing is called backward slicing/forward slicing, and the variables at some point of interest is called *slicing criterion*. The program slicing is what is called *static* slicing, which is computed without program execution. Dynamic slicing, which is based on program execution, is also proposed in [2] and some other papers, We apply only static slicing. In this paper, we use the word "slicing" as "static slicing".

Figure 1 shows a very simple example of program slicing. Many of readers might not be familiar with the syntax of COBOL, so we created the example in a Pascal-like language. The sample program originally gets two numbers and outputs the smaller one as `min` and the larger one as `max`, respectively. In this case, the variable `max` at the last line of the source code is the slicing criterion. The underlined statements are the static slicing computed for the slicing criterion. The slice shows the minimum statements required to compute the value of the variable `max` at the last statement. For example, in the case of program understanding for writing aspects, programmers should only read the underlined statements.

Figure 2 shows the data and control dependence relationships between the statements for the slicing, and control flow in the program. A dotted arrow means data dependence, and a dashed-two dotted line means control dependence. A solid arrow shows the control flow in the program. Only data and control dependence relationships related to the slicing in Figure 1 are shown here. Program slicing is computed by traversing data and control dependence. A graph that consists of data dependence and control dependence is called "Program Dependence Graph (PDG)" [8]. By traversing through PDG backward or forward, backward/forward slicing is computed. Figure 1 is an example of backward slicing.

2.2 Internal Control, Aspects and Program Understanding

As already described in Section 1, many world-wide companies have to make their information systems comply with the requirements of SOX-like laws [19]. For example, the compliance might require the addition of the following features to their information systems:

- Evidence log of business processing and transactions
- Data verification for finding invalid data (e.g., check digit)
- Comparison of the numbers of transactions between an order receiving system and billing system
- Access control of system users
- etc.

Implementing these additional features as aspects results in the separation of concerns for following business rules and concerns for strengthening internal control. Business rules are implemented in base programs as usual, and logic for strengthening internal control (e.g., logging) is implemented as aspects. In this case, when business rules are changed, the base programs will be modified, and when the SOX-like laws with which the information systems should comply are changed, the aspects will be modified.

In many companies, information systems written in COBOL process their business data (Aberdeen Group reported that 70% of the world's business data are processed in COBOL [1]). In order to realize the features by writing aspects, understanding COBOL programs is very important. Analyzing the existing COBOL programs is effective for the program understanding. In particular, program slicing with data flow analysis is very effective for program understanding. The SOX-like laws are about accounting in companies. For example, by computing program slices with respect to the *slicing criteria* that consist of money amount variables in input statements, only portions related to the internal control will be extracted. Programmers can design and implement aspects for the compliance only by reading sets of the statements extracted by slicing.

Also, when programmers develop aspects to improve internal control implemented in information systems, they might accidentally write some aspects that affect business logic implemented in base programs. By computing program slicing from joinpoints, portions potentially affected by the aspects can be extracted. H. Shinomi and T. Tamai presented the impact analysis of weaving in AspectJ in [15]. The program slicing is effective to prevent programmers from implementing aspects that cause unintentional changes to the base program.

3. ALTAIR: THE DEVELOPMENT ENVIRONMENT

Altair is a program analysis and development environment for COBOL programmers to apply aspect orientation to COBOL programs. Altair allows users to:

- understand base programs by analyzing structures and data flow (slicing)
- write COBOL aspects without a detailed knowledge of ALCOB by using a GUI

Figure 3 shows the relationships between Altair, Vega (aspect weaver), base programs, and aspects. Altair analyzes base programs and shows analysis results for the user. The user can understand the base programs by looking at the results shown through the GUI. By using aspect generation wizard of Altair, the user develops aspects and Altair generates aspects in ALCOB. The user can invoke the aspect weaver Vega through Altair, and Vega executes weaving of the base programs and the generated aspects. The user can see processing status of the weaving, and how the weaving was finally executed for the program. The aspect weaver Vega was implemented as a pre-compiler. Programs generated by Vega will be compiled by a COBOL compiler.

3.1 Language and Weaver

COBOL is a procedural programming language designed to address problems in business data processing. The latest standard COBOL 2002 [4] has object orientation syntax, but most enterprise COBOL users only use COBOL-85 syntax, which is a full procedural language. The base program on the right side of Figure 4 shows an example of COBOL compilation unit structure. The identification division contains program id and some other information. The data division contains data declarations that are used in the program. The procedure division contains the executable statements.

To begin, we added aspect-oriented features to the COBOL-85 syntax, because COBOL-85 syntax is widely used in enterprises. We firstly focused on realizing features needed to improve internal control (e.g., evidence logging) in enterprise information systems. ALCOB supports pointcut and advice features that are similar to AspectJ [18]. ALCOB supports *before* and *after* advice, but does not support *around* advice yet. AspectJ supports inter-type declarations, but ALCOB does not support inter-type declarations yet.

We have been analyzing many legacy programs written in COBOL. They are not necessarily written in exemplary coding style. For example, in many enterprise information systems, different data item names are used in different program units for same entity. These are called "synonyms". We added a construct (DEFINE statement) to ALCOB in order to absorb the difference and improve reusability of advice. The syntax of ALCOB is designed based on the experience of analyzing legacy programs. See [13] for the detailed language syntax and weaving semantics.

Figure 4 shows the overview of aspect-oriented construct and weaving semantics of ALCOB. The left side of the figure shows an aspect file of ALCOB. An aspect is a unit of compilation in the same way as a program is a compilation unit in COBOL. The aspect's name (or aspect-id) is declared at the beginning in the identification division. In the example, the aspect-id is `SALE-RECORD`. Next, there is the data division containing the working storage section, where data items (i.e., variables) are declared for use within the advice in the aspect. In this example, only one data item is shown, which is called `ASPECT-DATA-1`. Next, the procedure division contains the declarations of the advice in the aspect. In this example, only one advice declaration for `CALL-LOGGER` is shown.

Weaving semantics of ALCOB is also shown in Figure 4. The aspect weaver Vega reads aspect files and base programs, and extracts the context related to joinpoints. In the figure, the data items `USER-ID` and `CURRENT-SALE` are the context. Based on the pointcut declaration and the context, Vega selects portions where advice is to be inserted, generates the instance of the advice appropriate for the joinpoint, and finally inserts the instance to the portions. In this example, the instance of advice body (calling the logger program) of `CALL-LOGGER` is generated, and inserted before the following statement:

```
CALL "SALE-RECORDER" USING CURRENT-SALE
```

3.2 Program Understanding Support

As already described, COBOL programmers have to understand existing *legacy* programs for writing aspects. Altair has the following functions to support programmers to understand COBOL programs:

- Relation diagrams/lists generation (e.g., program caller-callee relationships, JCL (Job Control Language: a mainframe language for controlling batch jobs) and program relationships, etc.)
- Backward/forward program slicing
- Re-documentation, automatic document generation from source code (e.g., job flow diagram, program interface specification, file/record specification, record layout diagram)

Altair analyzes relationships between programs, variables, JCLs, and so on, and displays them in tree and list views. It supports the user to understand programs from several points of view. Figure 5 is a screen shot of part of the program caller-callee relation window of Altair. The relation is displayed hierarchically, and by clicking a program name in the tree view, the user can see detailed views (e.g., syntax-oriented source code browser) of the analysis results. Also, by selecting the program name, the user can automatically generate pointcuts based on the program name.

We already presented the importance of program slicing in Section 2. Altair supports backward and forward slicings of COBOL programs. By selecting data items (i.e., variables) at some statement of interest in syntax-oriented source code browser, the user

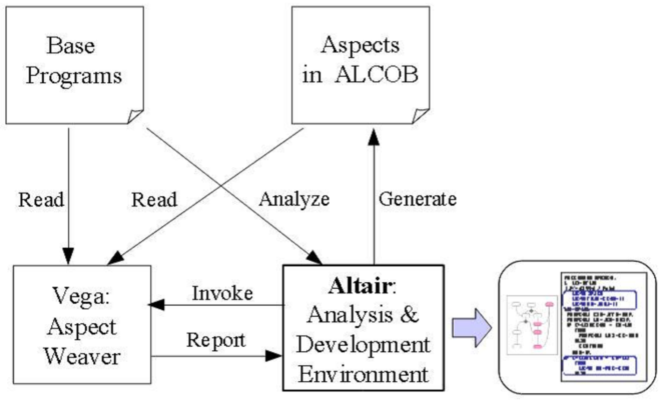
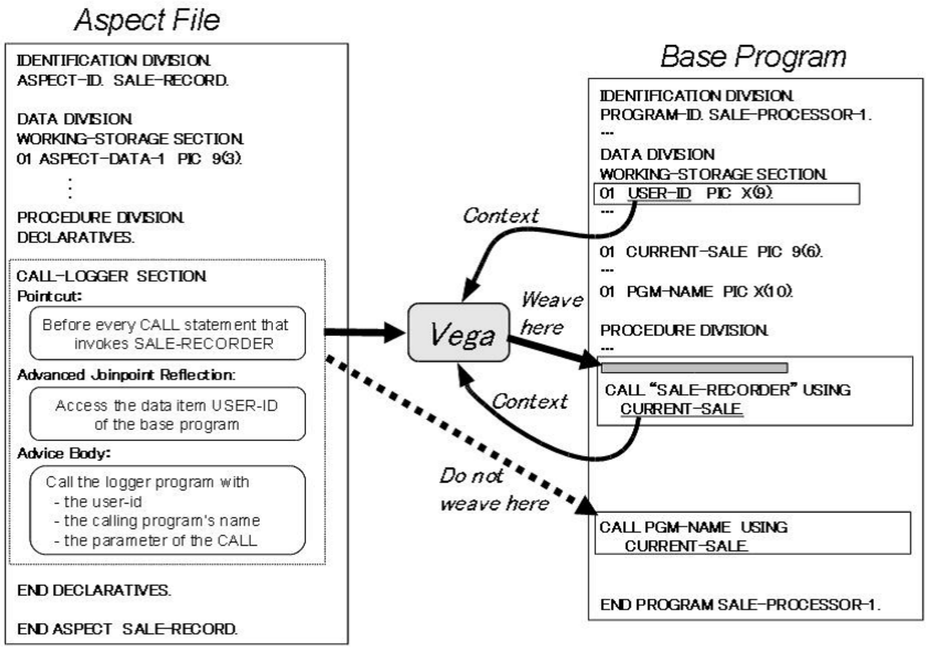


Figure 3: Altair and weaver



```
IDENTIFICATION DIVISION.
ASPECT-ID. SALE-RECORD.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 ASPECT-DATA-1  PIC 9(3).
        :
PROCEDURE DIVISION.
DECLARATIVES.

CALL-LOGGER SECTION.
Pointcut:
  Before every CALL statement that
    invokes SALE-RECORDER
Advanced Joinpoint Reflection:
  Access the data item USER-ID
    of the base program
Advice Body:
  Call the logger program with
   - the user-id
   - the calling program's name
   - the parameter of the CALL

END DECLARATIVES.

END ASPECT  SALE-RECORD.
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SALE-PROCESSOR-1.
---
DATA DIVISION
WORKING-STORAGE SECTION.
01  USER-ID   PIC  X(9).
---

01  CURRENT-SALE  PIC  9(6).
---
01  PGM-NAME  PIC X(10).

PROCEDURE DIVISION.
---

CALL "SALE-RECORDER" USING
  CURRENT-SALE.


CALL PGM-NAME  USING
    CURRENT-SALE.


END PROGRAM SALE-PROCESSOR-1.
```

Figure 4: Weaving example

can see set of indirectly relevant statements (e.g., underlined statements in Figure 1). The backward slicing results show the user the statements that affect the value of the selected variable, and the forward slicing results show the user the statements that are affected by the value of the selected variable.

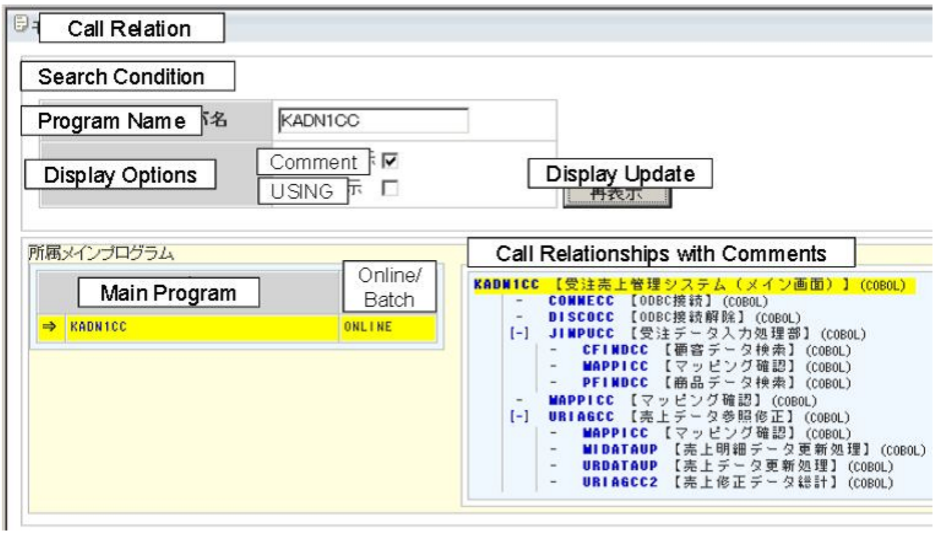
In the case of existing legacy systems, design documents are missing in a lot of companies. Even if design documents of the systems exist, designs described in the documents are frequently different from those implemented in the systems. Automatic document generation from source code is effective for understanding programs for writing aspects. Figure 6 shows a job flow diagram automatically generated by analyzing COBOL programs and JCLs. The diagram visualizes a batch execution, which basically consists of executable modules, input/output files, and database. In Figure 6, a box with two horizontal lines represents an executable module, and a cylinder represents data stored in a database. In the figure, the executable module inputs four kinds of data from a database and stores data to the database. A document box with "`SYSOUT`" represents data transfer to a dataset (i.e., file) through standard output. Altair automatically generates several kinds of documents, which are effective for program understanding and maintenance.

3.3 Aspect Writing Support

Altair supports the following functions for writing aspects:

- Wizard-based aspect generation
- Aspect management
- Invocation of the aspect weaver Vega
- GUI for displaying weaving status and results

In the program understanding views of Altair, the user can generate pointcuts or pointcut templates by selecting program name, statement, and so on (program elements). The phrase "pointcut template" means that the user can modify the generated pointcut



KADN1CC

Main Program	Online/ Batch
⇒ KADN1CC	ONLINE

```
KADN1CC 【受注売上管理システム（メイン画面）】(COBOL)
   -   CONNECC 【ODBC接続】(COBOL)
   -   DISCOCC 【ODBC接続解除】(COBOL)
  [-]  JINPUCC 【受注データ入力処理部】(COBOL)
     -   CFINDCC 【顧客データ検索】(COBOL)
     -   MAPPICC 【マッピング確認】(COBOL)
     -   PFINDCC 【商品データ検索】(COBOL)
   -   MAPPICC 【マッピング確認】(COBOL)
  [-]  URIAGCC 【売上データ参照修正】(COBOL)
     -   MAPPICC 【マッピング確認】(COBOL)
     -   MIDATAUP 【売上明細データ更新処理】(COBOL)
     -   URDATAUP 【売上データ更新処理】(COBOL)
     -   URIAGCC2 【売上修正データ総計】(COBOL)
```

Figure 5: Relation example: call relation

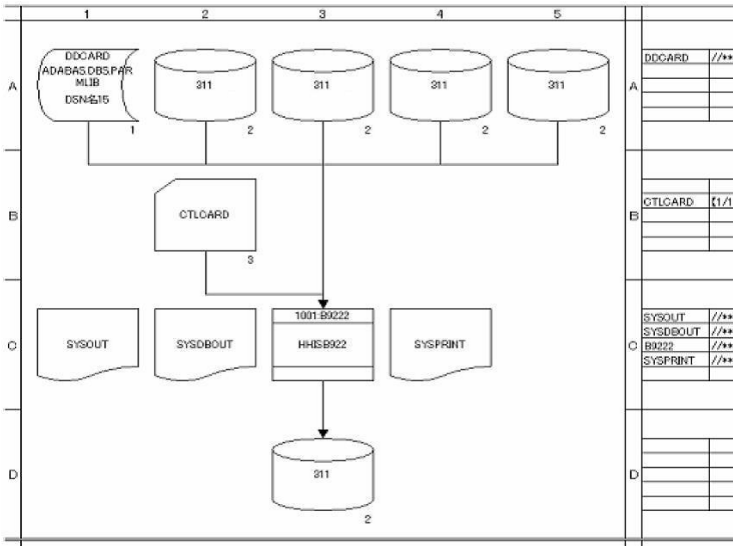


Figure 6: Re-documentation example: job flow diagram

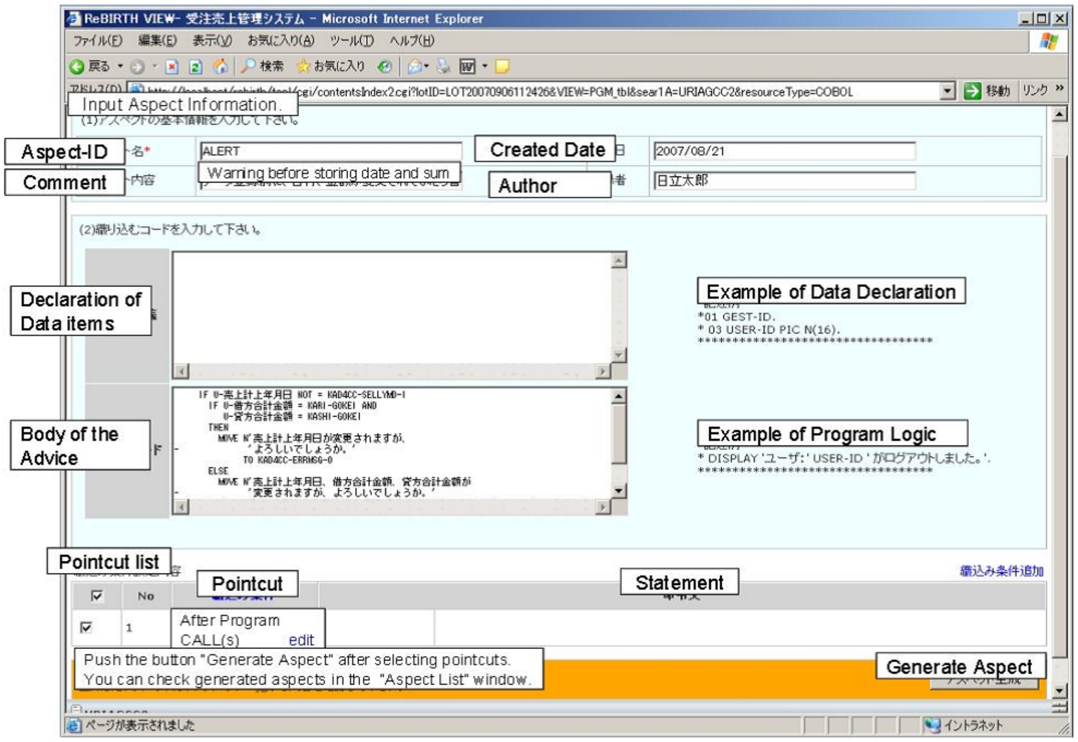
into more generic pointcut (e.g., by using the wildcard "*") through the GUI or a text editor. Altair can generate the pointcuts designating the following:

- inter/inner program routine call (CALL, PERFORM)
- file access statement (READ, WRITE, etc.)
- starting/end point of procedure (program, section, paragraph)
- statement (designated by line number)

By selecting one of them repetitively, several pointcuts or pointcut templates are generated for an aspect. Figure 7 shows the aspect writing wizard of Altair. Pointcuts for the aspect are listed at the bottom of the windows. In the pointcut list, the user can choose pointcuts that he or she wants to apply to the aspect. There are two text input fields in the middle of the window. They are for an advice declaration [13]. The top one is for declarations of data items (variables) that the advice uses, and the one below it is for COBOL statements to be the body of an advice. The user can input the two fields only with knowledge of COBOL. The pointcuts for the aspect are automatically generated based on the pointcut list. The user can create aspects without detailed knowledge of the aspect-oriented syntax of ALCOB (of course, he or she has to understand the concept of ALCOB). If the user wants to use the full specification of the ALCOB syntax, he or she might have to know the details of the syntax, but aspects with simple pointcuts and their combination can be created without detailed knowledge of the syntax.

Figure 8 is a screenshot of the aspect list window of Altair. All the aspects created in this project are listed in the window. Each aspect line has a check box at its left-hand side. The user can select aspects to be woven to the programs by checking the box. Also, by using the list, the user can open the window (Figure 7) for editing the aspect and delete some of aspects. There is a button "Invoke weaving" at the middle of the right-hand side of the window. The user can invoke the weaver Vega by clicking the button.

Figure 9 is a screenshot of the view of results of the invoked



ALERT

Warning before storing date and sum

2007/08/21

日立太郎

```
*01 GEST-ID.
* 03 USER-ID PIC N(16).
************************************
```

```
* DISPLAY 'ユーザ:' USER-ID 'がログアウトしました。'.
************************************
```

```
    IF U-売上計上年月日 NOT = KAD4CC-SELLYMD-I
      IF U-借方合計金額 = KARI-GOKEI AND
         U-貸方合計金額 = KASHI-GOKEI
      THEN
        MOVE N'売上計上年月日が変更されますが,
             'よろしいでしょうか。'
             TO KAD4CC-ERRMSG-O
      ELSE
        MOVE N'売上計上年月日、借方合計金額、貸方合計金額が
             '変更されますが、よろしいでしょうか。'
```

Push the button "Generate Aspect" after selecting pointcuts.
You can check generated aspects in the "Aspect List" window.

Figure 7: Aspect writing wizard

weaving. In this case, some aspects were woven into the listed programs `KAND1CC` and `URIAGCC`. The program `KAND1CC` has two portions where the weaving happened, and the program `URIAGCC` has three. By clicking the program name, the base program is displayed. In Figure 10, the statements marked (actually in yellow on the screen) means the portions where the weaving happened. There are two text strings "`after ALERT`" with a down-arrow and "`before CHECK`" with a up-arrow in the middle of the right-hand side of the figure. Those means that the after advice of the "`ALERT`" aspect and the before advice of the "`CHECK`" aspect are applied here to the two statements, respectively. By clicking the string "`after ALERT`", the window showing the definition of the aspect is popped up, and the user can check the details of the aspect.

4. TRIAL

We applied Altair to analyze a simple account information system and to strengthen internal control of the system by writing aspects. The simple account information system consists of 30 COBOL programs, and it has input/output screens and database access. It is a kind of order and sales management system. Users of the system can enter and change received orders and sales amounts through computer screens. The data is stored in a database.

The original system has the following problems from the standpoint of internal control:

1. The system did not display warning messages when the user entered or modified data (e.g., received order, sales amount).

2. The system did not output enough evidence logs for auditing.

In order to strength internal control of the system, we added the following features to the system by analyzing it and generating aspects:

1. Warning messages are displayed on the screen before the user updates data, and the update information messages are sent to his or her supervisor.

2. Evidence logs are output to a file (e.g., updated date, updated by whom, data before/after update, access history)

The first feature is "preventive control", and the second feature is "detective control" [6]. We created four aspects for implementing the features. One of them is shown at the right-hand side of Figure 11. In the aspect, the lines in gray are automatically generated by Altair. Only the lines in black are written by a programmer. The aspect added the following logic (advice) to the system:

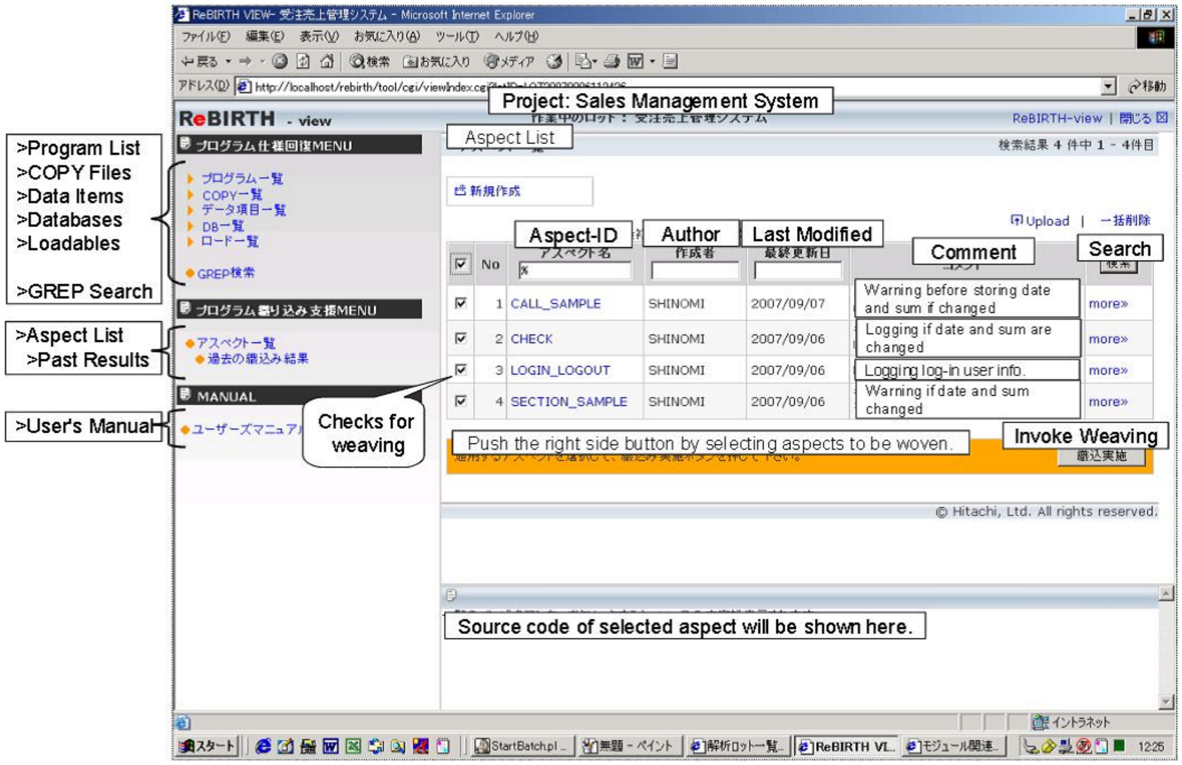
> If the sales date is updated, a warning message, which depends on whether the sales amount is also updated, will be displayed before calling the program `URIAGECC2`.

The left-hand side of the figure shows the change of the system between the screens before and after the weaving. The large arrow in the figure indicates that the text string assigned by the `MOVE` statement is displayed at the bottom of the lower screen. The COBOL `MOVE` is kind of like an assignment statement. For example,

```
MOVE A TO B
```

transfers data from `A` to `B`.

By using Altair, we successfully analyzed and generated the aspects, and successfully added some "internal control strengthening" features to the system. Current Altair supports only inner-program slicing. When we wanted to see any impact (slice) to other programs, we had to newly invoke the program slicing function from the affected parameters of the called program. This is a large limitation of the slicing (impact analysis) function. "Inter-program slicing" will be needed for analyzing more realistic information systems.



検索結果 4 件中 1 - 4件目

☑	No	アスペクト名	作成者	最終更新日	コメント	検索
☑	1	CALL_SAMPLE	SHINOMI	2007/09/07	Warning before storing date and sum if changed	more»
☑	2	CHECK	SHINOMI	2007/09/06	Logging if date and sum are changed	more»
☑	3	LOGIN_LOGOUT	SHINOMI	2007/09/06	Logging log-in user info.	more»
☑	4	SECTION_SAMPLE	SHINOMI	2007/09/06	Warning if date and sum changed	more»

Push the right side button by selecting aspects to be woven.

© Hitachi, Ltd. All rights reserved.

Figure 8: Aspect list window in the environment

5. RELATED WORK

AspectCobol was proposed by R. Lämmel and K. De Schutter. Their paper [11] introduced an aspect-oriented COBOL, but it does not describe its development environment like Altair.

As already described, ALCOB is another aspect-oriented COBOL proposed by T. Morioka, H. Danno, and H. Shinomi. ALCOB was implemented as the aspect weaver Vega. For detailed information of ALCOB and Vega, see the paper [13].

AJDT (AspectJ Development Tools) [3] is the most popular development environment for AspectJ [18]. AJDT is an Eclipse-based [7] integrated development environment, and it has excellent features for developing AspectJ programs. However, AJDT does not support a program slicing function.

H. Shinomi and T. Tamai [15][14] reported an Eclipse-based platform for analyzing Java and AspectJ, called JADE (Java and Aspectj DEpendence analyzer). A tool for impact analysis of weaving was developed on JADE, based on the program slicing technology. JADE might it easy to develop a program understanding tool like described in this paper for AspectJ.

T. Ishio et al. [9] proposed a debugging environment for AspectJ. Data dependences are analyzed by using execution time information. The slices are only on the path executed for specific input. This is not for analyzing programs statically for program understanding.

6. CONCLUDING REMARKS

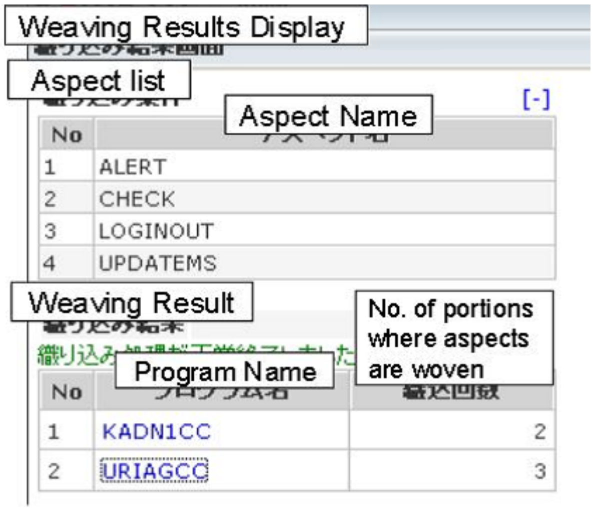
This paper described Altair, an environment for understanding and developing programs for aspect-oriented COBOL. There is an enormous amount of existing COBOL programs, and companies world-wide are going to be required to strength internal control implemented in their information systems. Aspect orientation can be applied to the solution. Existing COBOL program understanding and easy aspect writing are important to the application of aspect orientation. We realized them in the Altair environment. In particular, program slicing, supported by Altair, is effective for understanding existing programs and creating aspects to strengthen internal control of an information system.

We used Altair for implementing internal control features of a small information systems written in COBOL, and verified its effectiveness. It was verified that the woven code works as expected from the point of view of internal control. However, we found the limitation of the current inner-program slicing. Inter-program slicing might be required for more realistic information systems.

In addition to the described small sales management system, we applied Altair to source code of several information systems that are actually working in enterprises. By using Altair, we actually analyzed the source code and tried aspect weaving. We talked about the aspect-oriented approach to internal control with several systems engineers in Hitachi Ltd. They admired Altair's program analysis functions, but took a prudent attitude to the use of the woven code. They were weary of unintentional changes caused by weaving and the increase of testing workload of the woven code.

H. Shinomi, one of the authors, proposed the analysis of impact affected by weaving for AspectJ [15]. The algorithm of the impact analysis is also based on program slicing. By adding the similar function to Altair, the user might be able to check the unintentional changes caused by their aspects.

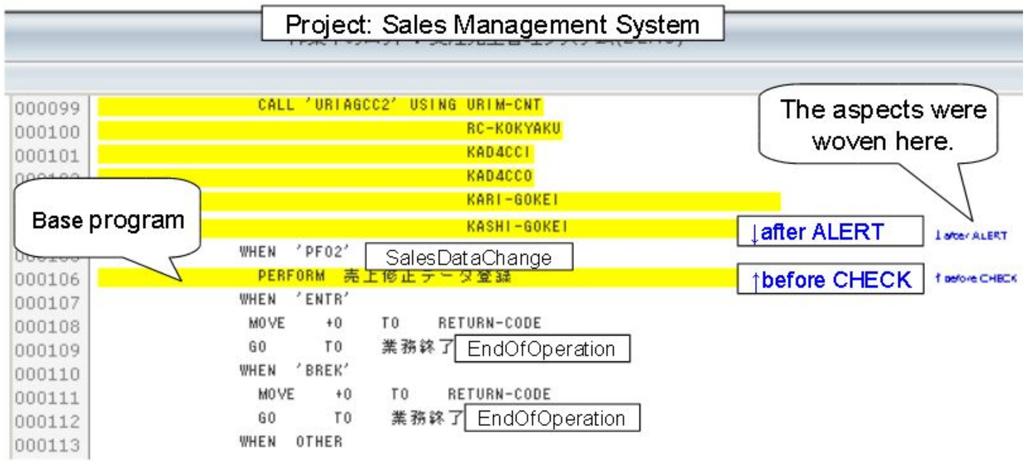
We think that the testing workload of manually modified code to improve internal control might not be different from that of the woven code. However, the impact analysis of weaving might be able to reduce the number of test cases to be performed. It can reduce the workload required to test the woven ode. The test case reduction based on the impact analysis was discussed in [16]. By enhancing the program analysis of Altair, a tool can be developed



No	Aspect Name
1	ALERT
2	CHECK
3	LOGINOUT
4	UPDATEMS

No	Program Name	No. of portions where aspects are woven
1	KADN1CC	2
2	URIAGCC	3

Figure 9: Weaving results



```
            CALL 'URIAGCC2' USING URIM-CNT
                                  RC-KOKYAKU
                                  KAD4CCI
                                  KAD4CCO
                                  KARI-GOKEI
                                  KASHI-GOKEI
          WHEN  'PF02'
            PERFORM  売上修正データ登録
          WHEN  'ENTR'
           MOVE    +0    TO    RETURN-CODE
           GO      TO    業務終了
          WHEN  'BREK'
            MOVE    +0    TO    RETURN-CODE
            GO      TO    業務終了
          WHEN  OTHER
```

Figure 10: Base program after weaving

for assisting programmers to reduce test cases.

Altair was developed as a web application having a browser-based user interface. We had already developed a COBOL program analysis tool when the idea of Vega and Altair occurred to us. So, we decided to develop Altair by enhancing the existing tool for quick implementation of the idea. Currently, however, Eclipse [7] is a de-facto standard for integrated development environment. Altair should be redeveloped on the Eclipse platform.

7. ACKNOWLEDGMENTS

The authors are grateful to Kouji Matsushita and Ryousuke Suzuki of Hitachi Systems and Services, Ltd. for his discussions and technical support of Altair. The authors are grateful also to Tsuyoshi Morioka and Hirofumi Dannno for their discussions of ALCOB.

8. REFERENCES

[1] Aberdeen Group, Inc. Legacy applications: From cost management to transformation, March 2003.

[2] H. Agrawal and J. R. Horgan. Dynamic program slicing. In *Proceedings of the ACM SIGPLAN '90 Conference on Programming Language Design and Implementation*, pages 246–256, White Plains, NY, June 1990.

[3] AJDT: AspectJ Development Tools. http://www.eclipse.org/ajdt/.

[4] COBOL Standards. http://www.cobolstandards.com/.

[5] COSO (Committee of Sponsoring Organizations of the Treadway Commission). Definitions - what is internal control? http://www.coso.org/resources.htm.

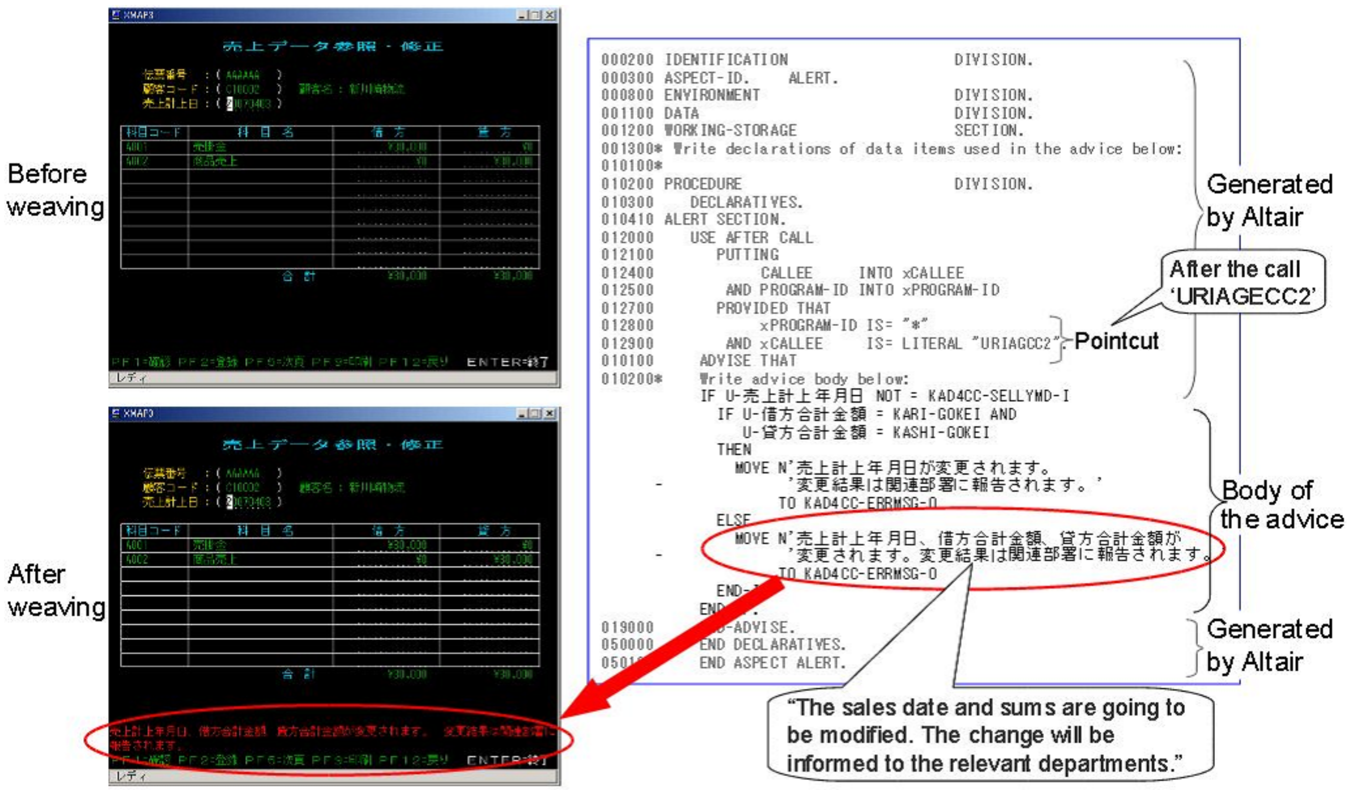
[6] COSO (Committee of Sponsoring Organizations of the Treadway Commission). Internal control - integrated framework. http://www.coso.org/IC-IntegratedFramework-summary.htm.

[7] Eclipse.org. http://www.eclipse.org/.

[8] S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. *ACM Trans. Program. Lang. Syst.*, 12(1):26–60, January 1990.

[9] T. Ishio, S. Kusumoto, and K. Inoue. Debugging support environment for aspect-oriented program – Using program slicing and call graph. *IPSJ Journal*, 45(6), June 2004.

[10] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proc. European Conference on Object-Oriented Programming*, pages 220–242, 1997.


Before weaving
After weaving
売上データ参照・修正
Generated by Altair
After the call 'URIAGECC2'
Pointcut
Body of the advice
Generated by Altair
"The sales date and sums are going to be modified. The change will be informed to the relevant departments."

```
000200 IDENTIFICATION                     DIVISION.
000300 ASPECT-ID.    ALERT.
000800 ENVIRONMENT                        DIVISION.
001100 DATA                               DIVISION.
001200 WORKING-STORAGE                    SECTION.
001300* Write declarations of data items used in the advice below:
010100*
010200 PROCEDURE                          DIVISION.
010300    DECLARATIVES.
010410 ALERT SECTION.
012000    USE AFTER CALL
012100       PUTTING
012400            CALLEE     INTO xCALLEE
012500        AND PROGRAM-ID INTO xPROGRAM-ID
012700       PROVIDED THAT
012800            xPROGRAM-ID IS= "*"
012900        AND xCALLEE     IS= LITERAL "URIAGCC2".
010100     ADVISE THAT
010200*    Write advice body below:
           IF U-売上計上年月日 NOT = KAD4CC-SELLYMD-I
             IF U-借方合計金額 = KARI-GOKEI AND
                U-貸方合計金額 = KASHI-GOKEI
             THEN
               MOVE N'売上計上年月日が変更されます。
      -              '変更結果は関連部署に報告されます。'
                  TO KAD4CC-ERRMSG-O
             ELSE
               MOVE N'売上計上年月日、借方合計金額、貸方合計金額が
      -              '変更されます。変更結果は関連部署に報告されます。
                  TO KAD4CC-ERRMSG-O
             END-
           END  .
019000     -ADVISE.
050000     END DECLARATIVES.
0501       END ASPECT ALERT.
```

Figure 11: Example of aspect

[11] R. Lämmel and K. De Schutter. What does aspect-oriented programming mean to Cobol? In *Proceedings of Aspect-Oriented Software Development (AOSD 2005)*. ACM Press, Mar. 2005.

[12] R. L. Mitchell. Cobol: Not dead yet: There are good reasons why cobol still runs many of the world's largest data centers. *COMPUTERWORLD*, october 2006. http://www.computerworld.com/s/article/266156/Cobol_Not_Dead_Yet.

[13] T. Morioka, H. Danno, and H. Shinomi. An aspect-oriented cobol for the industrial setting. In *7th International Conference on Aspect-Oriented Software Development (AOSD.08)*, Apr. 2008.

[14] H. Shinomi, H. Masuhara, , and T. Tamai. A tool for impact analysis of aspect weaving. In *International Conference on Aspect-Oriented Software Development (AOSD.06)*, Mar. 2006.

[15] H. Shinomi and T. Tamai. Impact analysis of weaving in aspect-oriented programming. In *Proc. the 21st IEEE International Conference on Software Maintenance (ICSM 2005)*, Sept. 2005.

[16] H. Shinomi and T. Tamai. Ripple effect analysis of aspect weaving. *Computer Software*, 23(3):170–188, 2006. (in Japanese).

[17] P. Tarr and H. Ossher. *Hyper/J User and Installation Manual*.

[18] The AspectJ project. http://www.eclipse.org/aspectj/.

[19] U.S. Goverment. Public Law 107 - 204 - Sarbanes-Oxley Act of 2002. http://www.gpo.gov/fdsys/pkg/PLAW-107publ204/content-detail.html.

[20] M. Weiser. Program slicing. *IEEE Trans. Software Engineering*, SE-10(4):352–357, July 1984.