

システムの機能強化の効率化のための アスペクト指向 COBOL

森岡 剛[†]

団野 博文[†]

四野見 秀明^{††}

多くの企業では COBOL で書かれた基幹システムが現在も稼動している。COBOL における横断的関心事のモジュール化を実現すること目的とし、COBOL 向けのアスペクト記述言語とアスペクトウィーバを開発した。プログラムの制御とデータ入出力を行う場面を主な織り込み対象とし、ポイントカット方式による織り込み箇所選定を行う。ジョインポイントの情報を参照するための仕組みとして文脈項目を定義した。ウィーバの織り込み性能評価を行い、実適用に耐える性能であることを確認した。

An Aspect-Oriented COBOL for Improved Productivity of System Enhancement

Tsuyoshi Morioka[†]

Hirofumi Danno[†]

Hideaki Shinomi^{††}

COBOL is still a vitally important language for mission-critical enterprise systems. In order to achieve improved modularization of cross-cutting concerns, we have developed an aspect description language and an aspect weaver for COBOL. Control structures of a program and loci of data I/O are included as joinpoints, and the pointcut is used as the means for selection of weaving points. As the mechanism for accessing the context of a joinpoint we defined the *context items*. A performance test confirmed that our weaver's weaving performance can meet the demand of the real-world setting.

1. はじめに

1.1. 産業界における COBOL の重要性

COBOL は 1960 年代に開発された「古い」言語であり、Java などの比較的新しい言語に比べるとその動向が注目を集めることは少ない。しかし、日本を含む世界中の多くの企業では COBOL で書かれた基幹システムが現在も稼動している。2006 年の Computerworld 誌の調査では、システム担当者の 62% が自社で COBOL を使用していると回答し、そのさらに 43% が、「自社開発のアプリケーションの 6 割以上は COBOL によるものである」と回答している[M06]。COBOL で書かれているアプリケーション(以下、COBOL アプリケーション)は今やメインフレームだけでなく PC やサーバなどのオープン環境でも稼動している。その「古い」イメージとは裏腹に、COBOL は今も重要な言語であり続けている。

COBOL 言語を用いた開発の効率化を実現するさらなる技術開発が望まれる。本稿は、株式会社日立製作所と、株式会社日立コンサルティング(以下、日立コンサルティング)による、アスペクト指向プログラミング(Aspect-Oriented Programming, 以下 AOP)技術の COBOL への導入の取組みを報告する。

1.2. アスペクト指向プログラミングと COBOL

AOP とは、関数やクラスにうまく局所化できない横断的関心事をアスペクトとしてモジュール化し、ベースプログラムに織り込む(to weave)ことでシステムを実現する技術である[K97]。本研究は、AOP 技術の適用によって COBOL 言語における横断的関心事のモジュール化を実現し、保守性と変更容易性を向上を実現することを目的とする。現在稼動中の COBOL アプリケーションの数と、その開発労力を考えれば、そのような技術のインパクトは大きい。

アスペクト指向 COBOL の先行研究としては、R. Lämmel と K. De Schutter の論文 [LDS05]がある。Lämmel らはアスペクト記述言語である *AspectCobol* を定義しその言語仕様開発上の指針について解説した。また、ウィーバのプロトタイプ実装を行った。さらに、一般的な横断的関心事の中には COBOL にはあまり関係ないものもあるが、ロギングや同期制御などは COBOL においても重要であると論じた。

1.3. 本研究の意義

本研究の主要な成果は、以下の二つである。

第一に、COBOL 向けのアスペクト記述言語の仕様を定義した。本稿では本言語を ALCOB (Aspect description Language for COBOL)と呼称する。

[†] 株式会社日立製作所 システム開発研究所
(Systems Development Laboratory, Hitachi, Ltd.)

^{††} 株式会社日立コンサルティング
(Hitachi Consulting Co., Ltd.)

第二に、ALCOB で記述したアスペクトを COBOL プログラムに織り込むための、アスペクトウィーバを実装した。以後、本ウィーバを「Vega(ベガ)」と呼称する。

ALCOB と Vega は実案件適用を念頭に開発したものであり、日立コンサルティングが既に提供しているソリューションの中核をなす技術である[H08]。本ソリューションについては2章で述べる。

本稿は ALCOB と Vega の概要を述べる。より詳細な情報源は[MDS08]がある。本稿の構成は以下のとおりである。まず第2章では ALCOB と Vega の仕様検討の方針を解説し、後に解説する仕様の意図を明らかにする。第3章では ALCOB 仕様の基本コンセプトを説明する。第4章は ALCOB 記法の概略を示す。第5章ではアドバイス織り込みの具体を解説する。第6章では Vega の実装と織り込み性能の評価結果について報告する。第7章は Vega をサポートする開発環境について報告する。第8章では ALCOB と Vega の特徴をまとめる。第9章では関連研究と関連技術について述べる。

2. 仕様検討の方針

本章では、ALCOB と Vega の仕様策定の方針を解説し、後に解説する仕様の意図を明らかにする。これらの方針は ALCOB と Vega の有用性とビジネスの価値を最大化することを目的として定めた。

方針(1): アスペクトの織り込みによる実行時の性能低下を極力避ける。 COBOL アプリケーションは企業の基幹業務を担うシステムの一部として稼動している場合が多い。業務への影響を最小限にするため、ALCOB と Vega の設計においては、織り込み自体が実行時の性能を低減させないように留意した。

方針(2): 特定の COBOL 製品ベンダーへの依存性を極小化する。 論文[LDS05]でも指摘されているように、産業的に有用なアスペクト指向 COBOL の検討においては、COBOL のコンパイラ、開発環境、実行環境などは複数のベンダーが存在する状況に留意する必要がある。

このような状況を鑑みて、Vega はソースコードレベルで動作するプリコンパイラ方式のウィーバとして実現することとした。つまり、Vega への入力は COBOL プログラム(ベースプログラムと呼称)とアスペクトであり、Vega の出力はアスペクトを織り込み済みの COBOL プログラムとした。さらに、アスペクト織り込み済み COBOL プログラムは、元のプログラムのコンパイルに用いていたコンパイラでコンパイルできるよう留意した。

方針(3): ALCOB がターゲットとする COBOL 標準は、COBOL-85 とする。 これは、アスペクト織り込みの対

象となるベースプログラムと、アスペクト織り込み済みプログラムの両方が、COBOL-85 準拠であることを意味する。COBOL 標準は複数存在するが、我々の経験によれば、現存する COBOL アプリケーションの大多数は COBOL-85 準拠の言語で書かれていると考えられる。

方針(4): アスペクト織り込み処理はビルドプロセスに組み込まれるものとする。 ウィーバが出力する織り込み済みソースはその後、人間が介入すること無しにコンパイルされ、リンクされ、デプロイされると想定する。織り込み後ソースを人間が修正することは想定しないが、アスペクトのデバッグなどの目的で織り込み後ソースを人間が解読する必要が生じることは想定する。織り込み処理はベースプログラムの構造を極力保存し、可読性を維持すること留意した。

方針(5): 織り込みを実現する目的でベースプログラムに修正を加えることは想定しない。 これは、ベースプログラムの修正無しにアスペクトを織り込むことが可能であり、既存システムを織り込み対象とすることができる。

上記方針(5)の帰結として、ALCOB はポイントカット方式を採用することとした。ポイントカット方式においては、アドバイスが織り込まれるべき箇所はポイントカットによって定義される。ポイントカットはアドバイスの一部として記述する。

ポイントカット方式以外の方式としては、ベースプログラム中に織り込みを制御するための情報を記述する、アノテーション方式がある。アノテーション方式が真にアスペクト指向と呼べるか否かは議論の余地があると言われる。アノテーション方式の実施例でよく知られるものとしては、Enterprise Java Bean (EJB)がある。

方針(6): ALCOB の記法は、可能な限り COBOL 的な記法とする。 これは、ALCOB と Vega のユーザとなる開発者の学習負荷軽減のためである。ALCOB が独自の記法を持つことは避けられないが、それが COBOL の自然な拡張と受け止められるよう留意して仕様策定を行った。

方針(7): ALCOB と Vega の設計と実装にあたっては、企業の内部統制強化への対応に必要な機能を優先して行う。 企業による財務報告に係る内部統制制度は、財務報告の信頼性を確保するための施策として2008年4月より実施が開始された[FSA07]。これは、財務報告に誤りを生じる可能性のあるリスクに対する統制(control)の有効性を検証することを企業に迫るものである。適用初年度である2008年度は業務の文書化などシステム外での対応がメインと言われる[N09]が、今後は統制の自動化による業務効率化、業務とシステムの

対応関係の明確化や、内部統制監査により露呈した不備への対応のため、システム改変を迫られるケースも多く出てくるものと考えられる。特に、既存システムの多くを占める COBOL アプリケーションに対し、ロギングやアクセス制御や入力チェックなどの機能をピンポイントで追加する必要が生ずると思われる。これらのニーズへの対応に必要な機能を優先し ALCOB と Vega の設計実装を行った。

3. ALCOB 仕様の基本コンセプト

本章では、AOP と ALCOB 仕様の基本コンセプトについて説明する。

3.1. アスペクトとアドバイス

AOPにおいては、プログラムに横断的に関係する関心事を、従来のプログラム構造と切り離れた単位で局所化する。その局所化の単位がアスペクトである。

ALCOB におけるアスペクトは1つ以上のアドバイスをから構成される。アドバイスはベースプログラムに挿入される COBOL コードとして表現される。

アスペクトがプログラム中の適切な場面で実行されるようにする役割を担うのがアスペクトウィーパーである。我々が開発した Vega は、COBOL プログラムとアスペクトを入力とし、適切な箇所にアドバイスを挿入した COBOL プログラムを出力する。

本研究は、AspectJ の考え方を踏襲し、AFTER、BEFORE、AROUND の3種のアドバイス[AJP]を考慮した。ALCOB はそのうちの BEFORE と AFTER の2種類のアドバイスをサポートしている。

3.2. ジョインポイント

アドバイスは、プログラム実行時の特定のタイミングにおいて実行される。そのような、アドバイスとの関連付けが可能なタイミングをジョインポイントと呼ぶ。ジョインポイントに対応するソースコードの構成要素(命令文など)をジョインポイントシャドウと呼ぶ[HH04]。本研究ではジョインポイントとそのシャドウを同一視し、「ベースプログラム中の、アドバイス織り込みが可能な点」と定義する。

COBOL プログラムの処理の記述を構成する最小の要素は文(命令文)であり、必要に応じて複数の文は段落(paragraph)や節(section)にグループ化される。このような COBOL の特徴を踏まえ、ALCOB におけるジョインポイントは以下の4種類とした。

- ①:「サブルーチン呼び出し」に関する文。これ CALL 文と PERFORM 文である。「サブルーチン」とは COBOL 言語には存在しない用語である。本

稿では便宜上、まとまった単位として呼び出し可能な命令文のグループを指し示す用語として使用する。

- ②:ファイル I/O に関する文。これは、OPEN 文、CLOSE 文、READ 文、WRITE 文、REWRITE 文、そして DELETE 文である。
- ③:サブルーチンとして呼び出し可能な構造。これはプログラム、節、段落である。
- ④:ソースプログラム中の行。

ALCOB も *AspectCobol* [LDS05]も命令文をジョインポイントとしているが、これは COBOL 言語における命令文の重要性を踏まえた仕様である。ALCOB と *AspectCobol* の比較は8章で行う。

サブルーチン呼び出しとファイル I/O は制御やデータの受け渡しが行われる箇所であり、統制上のリスクが顕在化しがちな箇所である。アスペクト織り込みによってロギング、アクセス制御、入力チェックなどの処理を追加するとの活用方法を想定し、ジョインポイントとして選定した。

行ジョインポイントは、COBOL プログラムのテキスト表現に存在するという点で、プログラムの論理構造上に存在する他のジョインポイントとは異なる。行ジョインポイントにより、プログラム中の任意の箇所にアスペクトを織り込むことが可能となるが、その乱用はアスペクトファイルを含むソースコードの保守をより困難にしてしまう。行ジョインポイントを用いた横断的関心事の織り込みは推奨しておらず、デバッグ目的や真にやむをえない場合の使用を想定している。

3.3. アドバイス

ALCOB におけるアドバイスの定義は以下の情報が必須である。

- ① ポイントカット:ジョインポイントのうち、該当アドバイスが織り込まれるべきジョインポイントを選択するための選定条件。アドバイスは、ポイントカットを満たす全てのジョインポイントに織り込まれる。
- ② アドバイスコード:アドバイスが実行する処理の記述。

必須でない情報としては以下がある。

- ③ ユーザ定義文脈項目のバインド:ジョインポイントに存在する情報のうち、ALCOB で既に定義されているもの以外を参照する際に必要。3. 7節で後述する。

3.4. アスペクトとアドバイスの構造

図1に、ALCOB におけるアスペクト定義の構造を模式的に示す。

図の左側はアスペクト定義の構造である。アスペクト定義は、見出し部(IDENTIFICATION DIVISION)、データ部(DATA DIVISION)、手続き部(PROCEDURE DIVISION)の3つの部から構成される。

見出し部にはアスペクト名 (ASPECT-ID)のみを記述する。図中では、SALE-RECORD がアスペクト名である。

データ部は、WORKING-STORAGE 節のみから構成される。ここでは、アドバイスが使用するデータ項目(変数)を宣言する。図では、データ項目 ASPECT-DATA-1 が示されている。ALCOB においては、データ部の記述は必須ではない。

手続き部は宣言部(DECLARATIVES)のみから構成される。宣言部には1つ以上のアドバイス宣言を記述する。それぞれのアドバイス宣言は節とする。図においては、CALL-LOGGER という名称のアドバイスの宣言が模式的に示されている。

CALL-LOGGER アドバイスを例として、ALCOB におけるアドバイスの概要を説明する。このアドバイスは、売上計上が起こるたびにその事実のログの出力処理を追加するために設計されたものである。売上計上は、SALE-RECORDER プログラムを呼び出すことで処理されると想定する。よって、CALL-LOGGER の織り込み箇所は、SALE-RECORDER を呼び出している箇所全

てである。この情報は、ポイントカットに記述する。ALCOB の記法では、ポイントカットは USE 文を用いて記述する。

CALL-LOGGER は、売上計上時に、売上計上処理を行ったユーザの ID、計上額と、織り込み箇所を含むプログラム名を呼び出したプログラムの名称をログとして出力する。CALL-LOGGER の処理は、アドバイスコードに記述する。実際には、CALL-LOGGER は、必要な情報を引数としてロギング用プログラムを呼び出すと想定する。アドバイスコードは、ADVISE 文を用いて記述する。

CALL-LOGGER はこのように、ユーザ ID、計上額、織り込み箇所を含むプログラムの名称という、ジョインポイントに存在する情報を取得し、それに基づいた処理を行わなければならない。ALCOB には、アドバイスがジョインポイントの情報にアクセスするための仕組みとして、文脈項目があり、ユーザ定義文脈項目はその一部である。文脈項目とユーザ定義文脈項目については後述する。CALL-LOGGER の例については、ユーザ定義文脈項目のバインドが必要であるが、それは DEFINE 文を用いて記述することのみ述べておく。

図1では、CALL-LOGGER アドバイスの表現は模式的に示されている。USE 文、DEFINE 文、ADVISE 文の実際の記法は、4章で説明する。

3.5. 文脈項目とポイントカット変数

ジョインポイントに存在する情報のうち、アドバイスからアクセス可能なものとして、ALCOB では文脈項目を定義する。文脈項目は、基本文脈項目とユーザ定義文

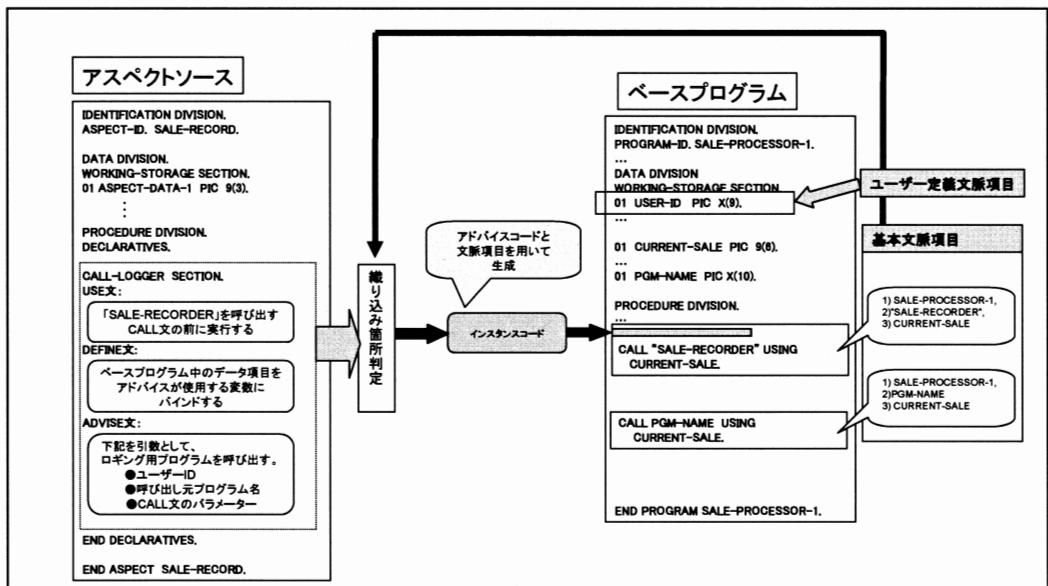


図1 ALCOB の概要

脈項目の2種類である。

文脈項目の用途は以下の2種類である：

用途①：ポイントカットにおいて、織り込み対象ジョインポイントが満たすべき条件を記述するために使用。

用途②：アドバイスコードにおいて、アドバイスが実行する処理を、個別の織り込み対象ジョインポイントに依存しない形式で記述するために使用。

用途①を、図1の CALL-LOGGER の例を用いて説明する。CALL-LOGGER の織り込み箇所は「SALE-RECORDERを呼び出すCALL文の前」であるが、そのようなCALL文を選別するには、各CALL文の呼び出し先(実際には、CALL文の最初のオペランド)を取得し、それが「SALE-RECORDER」という文字列と一致することを判定する必要がある。これは、3.6章で説明する基本文脈項目 CALLEE を用いて実現する。実際の USE 文の記述例は4章で説明する。

用途②の例として、CALL-LOGGER アドバイスのアドバイスコードの内容に着目する。ロギング用プログラムに渡す3つの引数はどれも織り込み対象のCALL文に存在する情報であり、アドバイス記述時点で記述することはできない。ADVISE 文の記述にはポイントカット変数を用いる。具体的な記述例は4章で示す。

3.6. 基本文脈項目

ALCOB におけるジョインポイントのそれぞれについて、基本文脈項目を定義した。基本文脈項目の幾つかは、全てのジョインポイントに共通である。例は、当該ジョインポイントを含むプログラムの名称である PROGRAM-ID やそのソースファイル名である SRC-FILE である。(ALCOB では、基本文脈項目を大文字で記述する。)

それぞれのジョインポイントには専用の基本文脈項目が定義されている。例えば、CALL 文ジョインポイントに対して定義されている基本文脈項目は以下のとおりである：

- (i) CALLEE: 当該 CALL 文が呼び出すプログラムの名称である。
- (ii) PARAM(n): 当該 CALL 文は呼び出し先プログラムに渡すパラメーターをオペランドとして持つが、PARAM(n)はその n 番目のパラメーターである。(n は1以上の整数。)
- (iii) SECTION-NAME と PARAGRAPH-NAME: それぞれ、当該 CALL 文を含む節と段落の名称。

- (iv) LINE-NUM: 当該 CALL 文の、ソースファイル中の行番号。

図1の右側に、プログラム SALE-PROCESSOR-1 の中に示されている2つのCALL文の1)PROGRAM-ID, 2)CALLEE, 3)PARAM(1)の値を示す。PROGRAM-ID は2つのCALL文で同じ値であり、SALE-PROCESSOR-1 である。CALLEE は、第一のCALL文は文字列 SALE-RECORDER であり、第二のCALL文ではデータ項目 PGM-NAME である。PARAM(1)は両方のCALL文で CURRENT-SALE となっている。

CALL-LOGGER アドバイスの織り込み箇所の判定は、CALLEE の値に基づいて行うことができる。すなわち、CALLEE の値が文字列「SALE-PROCESSOR」であるCALL文を織り込み対象ジョインポイントとする。

(ここで、図1の第二のCALL文のようにデータ項目が CALLEE であるCALL文の呼び出し先は、そのデータ項目の値によって動的に決定される。ALCOB においては、データ項目の値に基づく織り込みを行うことはできない。アドバイスコードにおいて、当該データ項目の値に基づいた処理を行うことは可能である。)

3.7. ユーザ定義文脈項目

アドバイスを設計する際に、ベースプログラム中に宣言されているデータ項目を参照したい場合があるが、一般にこれは基本文脈項目では対応できない。例えば、図1の CALL-LOGGER は、データ項目 USER-ID の値を参照する必要がある。

このような場合、ALCOB ではユーザ定義文脈項目を使用する。DEFINE 文を用いて、当該ジョインポイントをスコープとするデータ項目名などをポイントカット変数にバインドし、それをアドバイスコード中で使用することができる。

COBOL におけるデータ項目のスコープはプログラムであるので、同一データ項目が、プログラムごとに異なる名称を与えられる場合がある。ユーザ定義文脈項目は、このようなデータ項目名称の違いを吸収する機能も持つ。

3.8. 織り込みの概要

図1では、織り込み処理の概要も示している。まず、アドバイスとジョインポイントを確定し、織り込み箇所判定を行う。これは、基本文脈項目の値がポイントカットに記述された条件を満たしていることを判定することで行う。

図1では、第一のCALL文は満たしているが、第二

の CALL 文は満たしていない。よって、CALL-LOGGER の織り込み対象ジョインポイントとしては、第一の CALL 文のみが該当する。

次に、ベースプログラムに実際に挿入する COBOL コードを生成する。これを、インスタンスコードと呼ぶ。インスタンスコードはアドバイスコードを元に生成される。アドバイスコード中で使用されているポイントカット変数は、対応する文脈項目の値で置換する。つまり、インスタンスコードは、ジョインポイントの情報を加味して、アドバイスコードから生成される。

最後に、インスタンスコードをベースプログラムに挿入する。この部分の詳細は、5章で説明する。

4. ALCOB 記法の概要

本章では、ALCOB の記法を、図 1 の CALL-LOGGER アドバイスを例として説明する。紙面の都合上、概略の説明に留める。

4.1. USE 文

USE 文ではポイントカットを記述する。下に、CALL-LOGGER の USE 文を示す。

USE BEFORE CALL		
PUTTING	CALLEE PROGRAM-ID AND PARAM(1)	INTO xCALLEE INTO xPROGRAM-ID INTO xPARAM1
PROVIDED THAT xCALLEE IS = LITERAL "SALE-RECORDER".		

USE 文は 3 つの部分から構成される。まず、織り込み対象となるジョインポイントの種類と、アドバイスの種類 (BEFORE または AFTER) を冒頭で明示する。

次に PUTTING 句において、基本文脈項目を、ポイントカット変数にバインドする。この例においては、ポイントカット変数は、xCALLEE など基本文脈項目の名称に x を付けた名称としている。

最後に PROVIDED 句において、織り込み対象ジョインポイントを判別する条件を、ポイントカット変数を用いた論理式で記述する。

4.2. DEFINE 文

DEFINE 文では、ユーザ定義文脈項目のバインドを記述する。下に、CALL-LOGGER での例を示す。

この DEFINE 文は、ポイントカット変数 xUSER-ID のバインドを定義している。プログラム SALE-PROCESSOR-1 においては、データ項目 USER-ID とバインドする。SALE-PROCESSOR-2 においては、データ項目 USER-NAME とバインドする。そし

て、他の全てのプログラムにおいては、データ項目 UID とバインドする。

DEFINE xUSER-ID
IN PROGRAM "SALE-PROCESSOR-1" AS USER-ID
ALSO IN PROGRAM "SALE-PROCESSOR2" AS USER-NAME
ALSO IN OTHER PROGRAMS AS UID.

4.3. ADVISE 文

ADVISE 文は、アドバイスコードを記述する。CALL-LOGGER アドバイスの例を示す。

ADVISE THAT
MOVE xPROGRAM-ID TO ASPECT-DATA-1 CALL "LOGGER" USING ASPECT-DATA-1 xPARAM1 xUSER-ID
END-CALL
END-ADVISE.

「ADVISE THAT」と「END-ADVISE.」に囲まれた部分にアドバイスコードを記述する。基本的に COBOL の構文で記述可能である。大きな特徴は、ポイントカット変数を用いて記述することである。上記の例では、xPROGRAM-ID の値をデータ項目 ASPECT-DATA-1 に転記し、ASPECT-DATA-1 と xPARAM1 と xUSER-ID を引数としてロギング用プログラム (LOGGER) を呼び出す。

5. アドバイス織り込み

本章では、アドバイス織り込みを説明する。図 2 に、CALL-LOGGER アドバイスが図 1 の第一の CALL 文に織り込まれた結果を示す。1 つの織り込み箇所に対し、3 つの COBOL コードフラグメントが挿入される。

前述したように、インスタンスコード (図の(c)) は、アドバイスコードと文脈項目の値から生成する。インスタンスコードは新規の節として、ベースプログラムの手続き部の終端部に挿入する。

CALL-LOGGER アドバイスは BEFORE アドバイスであるので、ジョインポイントである CALL 文の実行前に(c)へ制御を移行しなければならない。これは、当該箇所に PEROFRM 文を挿入することで実現する (図の(b))。この PEROFRM 文は、(c)に制御を渡すサブルーチン呼出しとして理解可能であり、(c)の処理の終了時に制御は(b)に戻り、CALL 文が実行される。

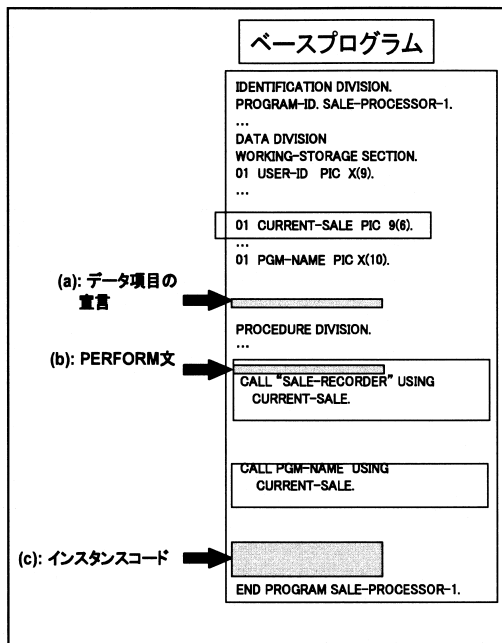


図2 アドバイス織り込み

CALL-LOGGER アドバイスコードではデータ項目 ASPECT-DATA-1 を使用しているが、その宣言はアスペクト定義の WORKING-STORAGE 節で行われている。このデータ項目はベースプログラムでは宣言されていないので、このままではコンパイルエラーを引き起こす。

コンパイルエラーを回避するため、ALCOB では、アスペクト定義に含まれるデータ項目宣言を、ベースプログラムの WORKING-STORAGE 節に挿入する (図2の(a))。

6. アスペクトウィーバ

Vega は、ALCOB を実案件で適用するために実装したアスペクトウィーバである。Vega は Windows® 環境で動作し、実装言語は C++ である。Vega は、日立コンサルティングが保有する COBOL 構文解析ライブラリを用いて実装された。

6.1. 織り込み性能のテスト

企業の基幹システムの一部を成す COBOL アプリケーションは大規模なものが多い。そのような場合でも Vega の織り込み性能が実適用に耐えるものであることを確認するため、テストを行った。

性能テストにおける Vega への入力とは以下のとおりである：(1) それぞれが 1 つのアドバイスを含む、2 本のアスペクトファイル。(2) 94 本の COBOL プログラム。合計ステップ数は約 200 K。(3) 113 本の COPY ファイル。これらは、合計 626 箇所を展開される。

2 つのアドバイスのうち、1 つは「名称が A で始まるプログラム」に対する BEFORE アドバイスで

あり、もう 1 つは特定のプログラムを呼び出す CALL 文への BEFORE アドバイスである。ベースプログラムと COPY ファイルは、実際の基幹システムのソースコードを使用した。

テスト環境は、2GB の RAM を搭載した Intel® Pentium® 4 3.60GHz プロセッサであり、OS は Windows Server® 2003 Standard Edition Service Pack 1 である。

テスト実行の結果、Vega の実行に要した時間は約 17 分であり、2 つのアスペクトの織り込み箇所は合計 164 箇所であった。実行時間の短縮が望まれるが、現状においても、現場での使い勝手としては許容範囲内と考えられる。

Vega の実行時間の内訳を分析したところ、ベースプログラムの構文解析に 60% 以上を費やしていることがわかった。ベースプログラムの構文解析が、Vega の性能に大きな位置を占めており、その性能改善は織り込み性能の改善に大きく寄与するであろうことがわかった。

7. 開発環境

既に述べたように、ALCOB 記法はできる限り COBOL に近いものとなるよう留意して定義した。しかし、AOP の概念は産業界、特に COBOL 開発者の間で広く普及しているとは言えない。そこで、ALCOB 記法と Vega のインターフェースの詳細を隠蔽し現場適用を促進するため、日立コンサルティングでは、ALCOB と Vega に対応した開発環境「Altair (アルテア)」を開発した。(Altair の開発には第一著者と第二著者は関与していない。)

ALCOB によるアスペクト設計作業においては、ベースプログラムの分析に基づいて必要なアスペクトを設計する作業と、適切な織り込み箇所をポイントカットに記述する作業の負荷が大きいのと思われる。Altair はこの部分の支援を行う。

開発者はまず、Altair を用いてベースプログラムの分析とアスペクト織り込み箇所の選定を行う。次に、Altair が表示する GUI ウィザードに従って必要な項目を入力することで、ポイントカットの記述ができる。アドバイスコードの記述は開発者自身が行う必要があるが、この部分はほとんど COBOL によるコーディングと変わらないため、開発者の負荷にはならないと思われる。

8. まとめ

本研究の目的は、現実の COBOL 開発への AOP 技術の導入である。そのため我々はアスペクト記述言語 ALCOB を定義し、ALCOB の実案件適用のためのアスペクトウィーバ Vega を開発した。

ALCOB と Vega の特徴は以下のとおりである。

(1) Vega は、ソースコードレベルで動作するウィーバであり、プリコンパイラの一つである。

(2) Vega を使用する際、ベースプログラムに対し、織り込み処理用にアノテーション等の新たな記述を追加する必要はない。

(3)COBOL で書かれた既存 COBOL アプリケーションの機能追加への対応を優先して設計実装された。

(4)ALCOB 記法は、COBOL にできるだけ近いものとした。

実案件適用を主眼として ALCOB と Vega の開発を行ったが、リソースの制約のため、未実現となった点もある。以下に、今後の改善点の幾つかを列挙する。

(1) ベースプログラム中で宣言され使用されるデータ項目、レコード、ファイルの分析の高度化。

(2) リレーショナルデータベースへのアクセス箇所へのアドバイス織り込み。

(3)COPY 文への対応。(COPY 文は、コンパイル時にコピーファイルの内容で置換される。COBOL においてはデータ項目定義を複数のプログラムで共通化するなどの目的で使用される。)

(4) 既存機能を差し替えるための、AROUND アドバイスの実現。

9. 関連研究と関連技術

9.1. AspectCobol と ALCOB の比較

COBOL のアスペクト指向化への取組みの先行研究としては、[LDS05]があり、本研究の基本的考え方は[LDS05]を部分的に踏襲している。具体的には、下記の部分である：(1) USE 文を用いたポイントカット記述。(2) アスペクト定義の全体構造。(3) アドバイスインスタンスを新規の節として挿入する点など、織り込みの基本的考え方。(4) 命令文、プログラム、節、段落をジョインポイントとする点。

[LDS05]と比較して、ALCOB の独自である点には以下がある。(1) ジョインポイントの情報をアドバイスから参照するための、文脈項目。(2) アドバイスがより広範な情報をジョインポイントから取得するための、ユーザ定義文脈項目と DEFINE 文。(3) アドバイスコード全体を1つの ADVISE 文として記述する点。

上記(2)の DEFINE 文の導入により、アドバイス宣言の再利用性が高まった。また、上記(3)の ADVISE 文の構造により、アスペクト定義の構造を簡素化することができた。ALCOB 仕様の簡素化と、アスペクトソースの可読性の向上に貢献している。

9.2. AspectJ と ALCOB の比較

ALCOB と AspectJ は、ポイントカット方式を用いている点は共通しているが、AspectJ は Java を対象とした AOP 言語であり、VM 上のバイトコードへのアスペクト織り込みを行う点で異なる。この方式は、Java と同等のプラットフォーム非依存性を実現できる利点と、織り込み後の再コンパイルが不要であるという利点がある。

COBOL の世界では複数のベンダが存在し、プラットフォーム非依存性を実現することは困難である。一つの対応策として、ALCOB と Vega は、COBOL-85 標準を対象としたプリコンパイラ方式とし、適用場面をできるだけ制限しないよう留意した。

また、Java プログラムの実行はメソッド実行の連続であるため、AspectJ のようにメソッド実行をジョインポイントとすることが自然である。ALCOB では類似の概念として、CALL 文や外 PERFORM 文によるサブルーチン呼び出しに関するポイントカットをサポートする。また、COBOL は基本的に命令文主体である点と、節・段落等のラベルを多用する点を踏まえ、これらのプログラム構造に関するポイントカット記述も可能とした。

謝辞

本研究の過程でご議論と技術的支援を頂いた、松下幸嗣氏と鈴木陵介氏(株式会社日立システムアンドサービス)そして一守康久氏(株式会社日立システムバリュー)に謝意を表する。

参考文献

- [AJP] The AspectJ Project, <http://www.eclipse.org/aspectj/>
- [FSA07] 金融庁企業会計審議会, 財務報告に係る内部統制の評価及び監査の基準並びに財務報告に係る内部統制の評価及び監査に関する実施基準の設定について, (2007) http://www.fsa.go.jp/singi/singi_kigyoutosin/20070215.pdf
- [H08] 株式会社日立製作所ニュースリリース, 2008 年 1 月 31 日.
- [HH04] E. Hilsdale and J. Hugunin, *Advice weaving in AspectJ*, Proc. of the 3rd international conference on Aspect-Oriented software development (AOSD '04), pp. 26-35 (2004)
- [K97] G. Kiczales et al., *Aspect-oriented programming*, Proc. European Conference on Object-Oriented Programming, pp. 220-242 (1997)
- [LDS05] R. Lämmel and K. De Schutter, *What does aspect-oriented programming mean to Cobol?*, Proc. of the 4th International conf. on Aspect-Oriented Software Development (AOSD'05), pp. 99-110 (2005)
- [M06] R. Mitchell, *Cobol: not dead yet*, Computerworld, October 4, 2006, <http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=266156>
- [MDS08] T. Morioka, H. Danno, and H. Shinomi, *An Aspect-Oriented Cobol for the Industrial Setting*, Industry Track Proceedings of the Seventh International Conference on Aspect-Oriented Software Development (AOSD.08), pp.77-87 (2008)
- [N09] J-SOX 追い込み ゴールは報告書の作成, 日経コンピュータ, 2009 年 1 月 1 日号, pp.84-93 (2009)