Ada向け構成管理の一考察

6S－7

梶原　清彦、山崎　誠一
ＮＴＴソフトウェア研究所

１．はじめに

複数のプログラマでプログラムを開発する場合には、お互いのモジュールの認識違いが開発を遅らせることになる。これを避けるためには,モジュールをプログラマ間で適切に構成を管理する必要がある。構成管理では以下の３レベルの管理がある。

・エディション
　対象となるプログラムに対する要求仕様に対応するレベル
・リビジョン
　対象となるプログラムのプログラム仕様に対応するレベル
・バージョン
　対象となるプログラムの実行環境に対応するレベル

本論文ではＡｄａでのリビジョン管理方式を検討する。なお検討にあたって以下の項目を前提条件とした。

* 多人数のプログラマによる開発であること
* 対象とするAdaコンパイラは複数のAdaプログラムライブラリ（下記の説明参照）の同時利用が可能であること
* 開発はUnix上で行なわれること

２．構成管理上のＡｄａの特徴

まず、本論文で対象としている言語「Ａｄａ」について、構成管理に関係する言語仕様とコンパイラを簡単に説明する。

(a) 仕様部と本体部

Ａｄａではプログラムを構成する各々のモジュールに対して仕様部（他のモジュールに見せる情報の定義）と本体部（モジュールの実体の定義）が存在する。

(b) Ａｄａプログラムライブラリ

Ａｄａではモジュール間のインタフェースチェックを行うためのデータベースであるＡｄａプログラムライブラリ（以後ライブラリを略記する）が存在する。

(c)コンパイル順序の存在

Ａｄａではモジュール間のインタフェースチェックを行うために、参照しているモジュールのコンパイルの前に、参照されるモジュールの仕様部のコンパイルが終了している必要がある。

３．Ａｄａにおけるリビジョン管理の問題点

まずリビジョン管理の一般的な問題点を挙げる。

・リビジョンの固定
　それぞれのプログラマが利用するモジュールのリビジョンが同一であること。つまり、全てのプログラマが利用するプログラム情報が同一であること。

・改変の通知
　プログラム情報がかわった場合、それを早く正確に他のプログラマに通知できること。

・効率的な開発
　プログラマが開発に集中できること。リビジョン管理に要する作業／時間が少ないこと。

次に開発言語がＡｄａであるがゆえの構成管理の問題点を説明する。

・ライブラリの存在　[(b)]
　プログラマが利用する情報はソースコードであるがＡｄａコンパイラが利用する情報はライブラリ内の情報であること。

・改変の影響　[(a),(c)]
　仕様部が変更されると、それを直接／間接に参照しているモジュールはコンパイルし直さなければならない。

４．リビジョン管理に要求される項目

リビジョン管理方法を考えるにあたって、考慮すべき点を以下にリストアップする。

(1)管理する対象

管理すべき情報は多すぎてはいけない。また管理している情報の状況を正確に把握できなければならない。

(2)リビジョンの固定方式

確実にリビジョンを固定する上で、管理し易く、ミスの可能性が小さい管理でなければならない。

(3)リビジョンアップ手順

管理する側にもプログラマにも、あまり負担をかけず、また手順を間違えにくいリビジョンアップ手順でなければならい。

(4)改変した情報の通知方式

他のプログラマに関連するモジュールを修正した場合、早く正確に管理者及び各々のプログラマに通知しなければならない。

(5)効率化

各々のプログラマはプログラミングに集中できなければならに。また他のプログラマと非同期的に作業できなければならい。

(6)工程によるリビジョン管理の差異

リビジョン管理を行なわなければならないのは、コーディング工程、単体試験工程、結合試験　工程、保守工程である。それぞれの工程における構成管理の推移が容易に行えなければならない。

5. リビジョン管理方式の提案

3節で説明したリビジョン管理の問題点を踏まえ、筆者が考案したリビジョン管理方式を説明する。ただし、紙面の制限のため開発工程による差異には触れず、基本的な考え方のみを説明することとする。

ソースコードとライブラリの両方を管理する。一貫性が保たれいている共用ソースコードと対応するライブラリ（以後、共用ライブラリと記す）と各々のプログラマの最新ソースコードと対応するライブラリ（以後、最新ライブラリと記す）を置く。共用ライブラリと共用ソースコードは管理者が管理し、最新のソースコードは各々のプログラマが自由に変更できるようにするが（当然、コンパイル可能なものであり、最新ライブラリに登録されるとともに、既存の構成管理ツールRCSなどによってリビジョンを保存する）、管理者が監視する。

プログラマはプログラマの状態に対応した3つのライブラリを用いる（図1参照）。コーディングが遅れている状態では、認知されている一貫性のある最新リビジョンである共用ライブラリとリンクされている個人ライブラリを用いる。プログラマが特定の状況に興味がある状態では（例えばバクの解析）、必要な全てのソースコードをコンパイルしたワークライブラリを用いる。コーディングが進んでいる状態では最新の状況に対応するために、テストライブラリを用いる。これは最新ソースコードが変化するたびに更新される最新ライブラリにリンクされた（そのために最新ライブラリが更新される度に自動的に更新されるようにする）ライブラリでる。

管理者は適切なタイミングで最新ソースコードを共用ソースコードとし、共用ライブラリにコンパイルする。もちろん、一貫性がとれていなければならない。

6. 提案した方式の検証

ソースコードとＡｄａライブラリ両方を管理する方法は共用ライブラリ及び最新ライブラリの状況を正確に把握できるとともに、各々のプログラマは共用ライブラリを参照することで、リビジョンの固定を図ることができる［(1),(2)］。さらに、プログラマはそれらのソースコードをコンパイルする必要がない［(3)］。当然、各々のプログラマは自分専用のライブラリを利用するので、他のプログラマと同期をとって開発する必要はない［(5)］。

プログラマは共用ライブラリまたは最新ライブラリを参照するだけなので、プログラマ側からのミスはほとんどない［(3)］。管理者のミスを防ぐ方法が必要であるが、自動化が可能であればミスは少ない。

通知手段に関しては、テストライブラリによって、何がどう変わって、自分に影響があるのかがすぐに判る［(4)］。

バグの解析時などでも、ワークライブラリによって他のプログラマの影響を受けない［(5)］。またコンパイル順序が存在するので、参照している先の仕様部が頻繁に変わる場合でも個人ライブラリによって作業を進めることができる［(5)］。

7. 今後の検討課題

・管理作業の自動化について

シェルスクリプトの作成、atコマンド／cronデーモンの利用によって、共通ソース／共通ライブラリの作成、改変の通知、プログラマのライブラリの再リンクなどを自動化／半自動化することが可能である。何を、どのようにすれば、自動化できるのか、検討する必要がある。

・ＬＡＮへの対応について

Unixワークステーションで開発を行なう場合、ＬＡＮで結合された複数台のマシンでコーディング、デバックが多い。ネットワークファイルシステムによる効果的な構成管理を行うためには、どのようなネットワークファイルシステムを構築したらよいのか検討する必要がある。

8. まとめ

多数のプログラマが並行してプログラムの開発を行ない、ソースコード以外にもライブラリが存在するという複雑な環境でのリビジョン管理に対して、適切な方式を考えた。この方式を実際に適用するには、それぞれの環境に即した方式の細部の検討が必要があるので、現在検討をおこなている。今後、この方式を実際のプロジェクトに適用し、方式の改善を目指すとともに、自動化ツールの作成を行なっていきたい。

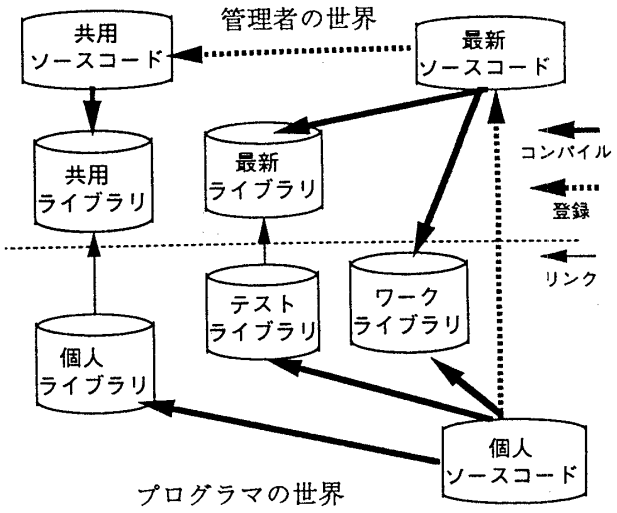

図1　ライブラリの構成

・注意

・「3．Ａｄａにおけるリビジョン管理の問題点」のAdaに関する問題点の項目名の最後のカギ括弧内の数字は、「2．構成管理上のＡｄａの特徴」の項目記号と対応する。

・一貫性とは、全てのモジュールのソースコードがエラーなくコンパイルできることである。

・「6．提案した方式の検討」の文の最後のカギ括弧ないの数字は、「4．考慮すべき事項」の項目番号と対応する。