ソフトウェア工学 79－3
（１９９１．６．２５）

共同開発時におけるインテグレーション支援機能

池田健次郎、坪谷英昭、岸知二

日本電気 (株) ソフトウェア生産技術開発本部

ソフトウェア開発支援環境は、開発過程で生み出される成果物を管理するための機能を提供しなければならない。我々は既に、ソフトウェア開発環境を構築する際のプラットフォームとして、成果物や設計情報を管理するためのライブラリ LifeLine を開発している。また LifeLine を用い、成果物管理機能、構成管理、版管理の機能および、ロードモジュールの生成といったインテグレーションの支援機能を提供する C プログラム開発環境 LifeStudio/C を開発した。本稿では、共同開発時におけるインテグレーション支援に注目し、LifeStudio/C の中でインテグレーション支援機能がどの様にして実現されているか報告する。

Derived Objects Management for Programming-in-the-Many

Kenjiroh IKEDA　Hideaki TSUBOTANI　Tomoji KISHI

Software Engineering Development Laboratory, NEC Corporation

Igarash building, 11-5, Shibaura 2-chome, Minato-ku, Tokyo, 108, JAPAN

A software development environment must present facilities to manage artifacts which are produced in software develpment process. We have developed LifeLine library for managing artifacts and design information, as a basis for constructing software development environments. And we have developed LifeStudio/C, a software development environment for C programs, which is constructed using LifeLine and provides such facilities as configuration management and version control . In this paper, we present LifeStudio/C and the management mechanism for derived objects.

1 はじめに

近年、大規模なソフトウェアを複数のメンバで開発することが多くなってきた。そして、このような開発過程で生み出される様々な成果物を効果的に管理していくことが重要な課題となってきている。

成果物管理において生じる問題を解決するために、我々はソフト開発環境を構築する際のプラットフォームとして、成果物や設計情報を管理するための機能 LifeLine を開発した [1]。LifeLine は、UNIX 上の種々の CASE ツールに組み込まれて使用されることを目的とした C ライブラリである。

LifeLine で管理される対象となる成果物は大きく二種類に分けることができる。一つは作業者がエディタ等を用いて直接作成するものであり、各種ドキュメント、図式、ソースコード、等がこれに含まれる。もう一つは他の成果物から機械的な手続きによって生成できるものであり、オブジェクトコードやロードモジュール等はこれに含まれる。LifeLine では前者を単にファイル、後者をターゲットファイルと呼ぶ。

また LifeLine を用いて C プログラムの開発環境として LifeStudio/C を開発した [2]。LifeStudio/C では、成果物の一元管理に加え、構成の管理、版管理の機能を提供するとともに、ターゲットファイルの作成といったインテグレーション支援機能を提供する。ここで、インテグレーション支援機能とは、ターゲットファイルを生成する時のソースとなるファイル、生成方法、格納場所を作業者の代りに管理し、ターゲットファイル生成の支援をする機能である。

ロードモジュール等のターゲットファイルを生成作業を支援するものとしては、Make [3] がよく知られており、それを機能拡張した NMAKE [4] 等のツールや、DSEE [5] 等のシステムがある。LifeStudio/C では UNIX との融合性を考え、内部で Make を呼び出す形でインテグレーション支援機能を実現している。

本稿では、LifeLine における情報管理の考え方を示しながら、LifeStudio/C の提供する機能の一つ、インテグレーション支援機能について述べる。なお、OS として UNIX を想定している。

2 従来のプロセス

大規模なソフトウェアを開発する場合、いくつものモジュールに分割し、複数の人で開発を行う。このような開発のしかたをする場合、次のような問題が発生する。

1. 同一のファイルがコピーされて二箇所以上の場所で管理され、それらの間で情報の整合性がとれなくなる (二重保守問題)。
2. 一人の作業者が行ったファイルの修正が他の人へ影響を及ぼす (共有データ問題)。

これらの問題を解決するためには、共有のプロジェクトデータベースとしてのベースラインと、作業者毎のワークスペースとの組み合せによる他制御を行えばよい。従来は、このベースラインとワークスペースをディレクトリにより表現していた。

例えば、ロードモジュールの作成の場合には以下のような手順になる。

1. 作業者がソースファイルの修正を行う場合、成果物が一元管理されているベースラインとしてのプロジェクトディレクトリから、必要なソースファイルを自分のワークスペースとしての作業用ディレクトリにコピーして修正する。
2. その修正結果を確認するためにコンパイル・リンクといったインテグレーション作業を行なうが、その際にどのソースファイルを参照するかは作業者が特定する。
3. インテグレーション作業が終り、修正の結果を正しく確認できればソースファイルを元のプロジェクトディレクトリにコピーして戻す。
4. 全てのソースファイルの修正・動作確認が終ると、ベースラインにおいて全メンバの作業結果を反映した最終的なインテグレーション作業が行われる。

3 問題

従来の様な開発フローでは、インテグレーション作業において次のような問題が生じてくる。

3.1 元となる成果物の特定

インテグレーション作業を行う場合、作業者はソースファイルの様な元となるファイルを特定しなくてはならない。

ベースラインにおける最終的なインテグレーション作業の場合には、ベースラインにあるファイルを単純に参照すればよい。しかし、作業者がワークスペースにおいて修正したファイルの動作確認を行うためにインテグレーション作業をする場合には、ベースラインからコピーしてきて修正したファイルと、それ以外のベースラインにあるファイルを参照しなければならない。

ところが、開発するソフトウェアが大規模化して分割するモジュールが多くなったり、ベースラインとワークスペース間のやりとりが多くなってくると、どのファイルを参照すれば良いかといった管理が繁雑になり、インテグレーション作業の妨げとなる。

そこで、様々な開発の状況においてインテグレーション作業に必要なファイルを間違いなく特定する機能が必要となる。

3.2 生成方法の特定

作業者がインテグレーション作業を行う場合、生成されるべきターゲットファイルが何であるかによって、ターゲットファイルの生成方法が変わってくる。ロードモジュールを生成する場合にはロードモジュール用の、ライブラリを生成する場合にはライブラリ用の生成方法を選択しなくてはならない。

また、複数のマシン機種に対して開発を行う場合、マシンの機種によって生成方法が変わってくる場合がある。

この様にインテグレーション作業を行う場合には、生成されるターゲットファイルの種類や対象となるマシン機種により、ターゲットファイルの生成方法を間違いなく特定する機能が必要になってくる。

3.3 格納場所の特定

単一のマシン機種に対してソフトウェア開発を行う場合は、生成されるオブジェクトコード等のターゲットファイルの格納場所には特に気を使わなくてもよい。ところが、複数のマシン機種に対して同時に開発を行っていく場合には、ターゲットファイルの格納場所の管理が必要となってくる。

格納場所を管理する方法の一つとして、マシン機種毎にベースラインと同じ構造のディレクトリを用意し、ソースとなるファイルにシンボリックリンクを張ることによりターゲットファイルの格納場所を分ける方法がある。しかしこの方法では、ベースラインの構造が変わるとそれにあわせてディレクトリ構造を変えたり、シンボリックリンクを張り直したりしなくてはならず、管理が繁雑である。

そこで、より簡単にターゲットファイルの格納場所を特定し、ターゲットファイルを管理する機能が必要になってくる。

4 LifeLine が提供する機能

LifeLine ライブラリは成果物としてのファイルと、それに関連する設計情報を管理する機能を持っている。LifeLine では、これらを管理するためにプロジェクト、フォルダ、ファイル、コンポジション、ターゲットというものを用意している。

プロジェクトは成果物や設計情報の管理空間である。全ての情報はプロジェクト中で管理される。プロジェクト中ではフォルダによって階層的な管理構造が定義される。フォルダは UNIX のディレクトリと 1 対 1 に対応している。フォルダ中には成果物であるファイル、構成を表すコンポジション、ロードモジュールの様に他のファイルから生成されるファイルを表すターゲット等が管理される。また LifeLine では、フォルダ、ファイル、コンポジション、ターゲットをあわせてオブジェクトと呼ぶ。

4.1 プロジェクト階層

LifeLine では、管理すべき成果物は、プロジェクトと呼ばれる論理的な単位にまとめられる。プロジェクトはプロジェクトディレクトリと呼ばれるディレクトリと一対一に対応付けられる。プロジェクトにはルートプロジェクトとサブプロジェクトの二種類があり、これらによりベースラインとワークスペースを実現している。

ルートプロジェクトはベースラインに相当し、ある開発作業に対して一つ作成される。その開発作業において管理される成果物は、最終的にはそのルートプロジェクト中にまとめられる。排他制御の機能を用いる必要がないときには、このルートプロジェクトだけで作業を行うことができる。

サブプロジェクトはワークスペースに相当し、ルートプロジェクトあるいはその他のサブプロジェクトの子供として生成される。すなわち、ある開発作業に関連するプロジェクトは、ルートプロジェクトを起点とする木構造を構成することになる。この木構造をプロジェクトツリーと呼ぶ。サブプロジェクトは一時的な作業エリアであり、排他制御を実現する必要がある際には作業者毎にサブプロジェクトを生成する。この様にプロジェクトツリーは開発作業の階層を表現することになる。

4.2 コンポジション

成果物の管理においては、その構成情報を管理する必要がある。一般に構成は成果物の管理構造とは異なることが多く、例えばファイルシステムの提供するディレクトリ構造などを用いて構成を表現することは、一般に困難であるといわれている。LifeLine では管理構造を表現するフォルダとは別に、構成を表現する手段としてコンポジションを用意している。コンポジションは、ファイル、ターゲット、他のコンポジションの名前のリストであり、フォルダによる管理構造とは独立に、階層的に構成を表現することができる。コンポジションにリストの項目として登録されているオブジェクトを特にコンポジションエレメントと呼ぶ。

例えば、図 1(a) のようなフォルダ構造が作られており、フォルダ A、B、C 中にコンポジション CompA1、CompA2、CompB、CompC が存在している場合には、図 1(b) に示すように 3 つのコンポジション構造が存在している。

また、コンポジションは型を持つ。コンポジションの型はそのコンポジションが含むことのできるファイルのファイル型の集合として定義される。そして、コンポジションに対する基本操作の一つとして、コンポジション中に含まれるファイルから特定の型のファイルのみを取り出すことができる。

4.3 リザーブ・デポジット・二重化

LifeLine が支援する機能の一つとして、複数人の共同開発時における情報のアクセス制御の機能がある。これはベースラインとワークスペースでの情報のやり取りによって解決する。すなわち作業者はベースライン中にある成果物等にロックをかけ、それを自分のワークスペースにコピーしてその修正を行なう。その間他の作業者はその成果物に対して修正作業を行なうことはできない。修正作業が終了すると、作業者はその成果物をベースラインに返し、ロックを解除する。LifeLine ではこの操作をリザーブ・デポジットと呼ぶ。リザーブ・デポジットは、フォルダ、ファイル、コンポジション、ターゲットに対して行なうことができる。

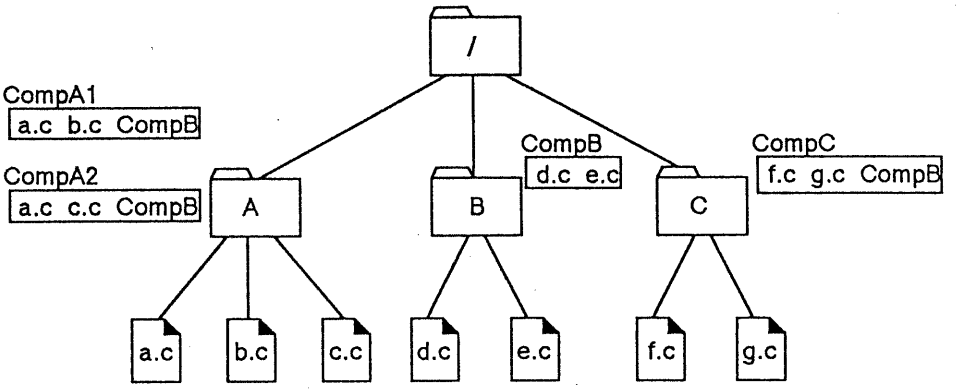
LifeLine では、リザーブの他に二重化という機能を提供している。二重化はオブジェクトの修正権をとらずに、オブジェクトのコピーだけをカレントプロジェクト中に作成する処理である。二重化は自分だけが参照する目的で過去の版をチェックアウトしたり、コンパイルを行なったりする際に用いられる。二重化されたオブジェクトに対しては、リザーブしたオブジェクトと同様に修正作業を行なうことができるが、その結果を親プロジェクトに返すことはできない。二重化している間に、親プロジェクト中の該当オブジェクトが修正されても、あくまでカレントプロジェクト中のオブジェクトの内容しか見ることができない。またあるプロジェクトが二重化を行なうと、その子プロジェクトは二重化されたオブジェクトしか見ることができなくなる。

5 インテグレーションのメカニズム

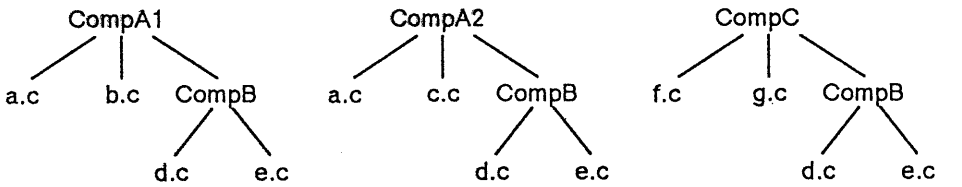
LifeStudio/C は、前述した LifeLine の機能を用いて C プログラムの開発支援環境を構築している。ここでは、インテグレーション支援機能がどうのようなメカニズムで実現されているか述べる。

5.1 コンポジションエレメント

インテグレーション作業はコンポジション単位で行われる。そこで、インテグレーション作業に必要なファ



(a)フォルダ構造



(b)コンポジション構造

図1 フォルダ構造とコンポジション構造

イルはコンポジションエレメントとして登録することにより、その構成を管理することができる。

またLifeLineはアクセス制御機能を実現するため、コンポジションはそのエレメントがどの様な状態にあるかという情報をあわせ持っている。つまり、コンポジションエレメントを参照する場合、どのプロジェクト階層にあるものを参照すれば良いか一意に定めることができる。

図2(a)の様なコンポジション構造がある場合、初期状態においてサブプロジェクトではどのファイルもリザーブしていないので、各ファイルの参照先はルートプロジェクトにあるファイルになっている。ここで、d.cをリザーブすると、ルートプロジェクトにあるd.cのコピーがサブプロジェクトに作られ、ファイルのリザーブ・デポジットの状態の変更に伴い、サブプロジェクトにおけるd.cの参照先もルートプロジェクトからサブプロジェクトに切り替えられる(図2(b))。

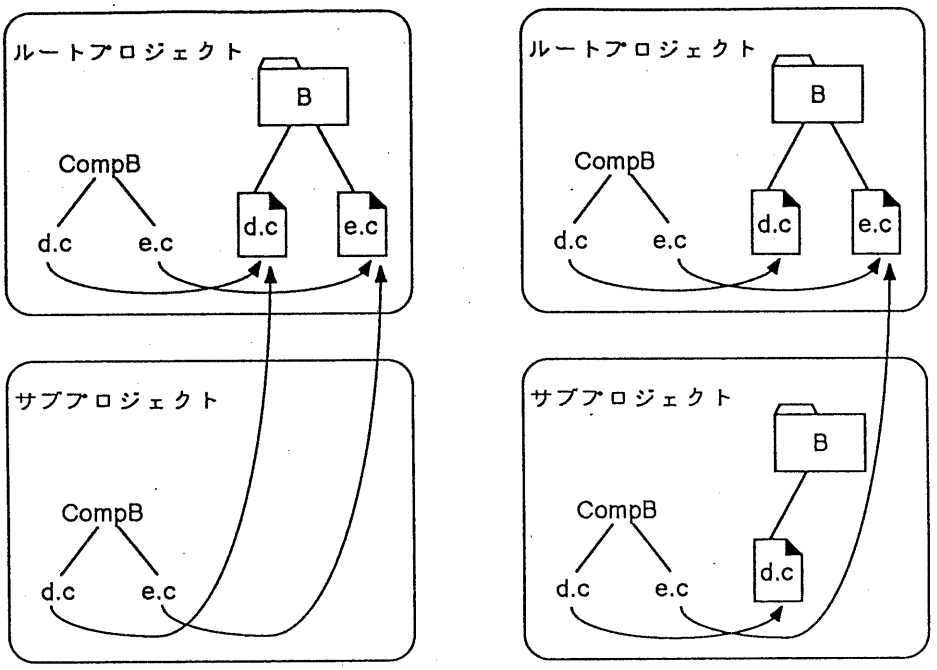
この様に、常にコンポジションはエレメントの参照先の情報を正しく保持しているので、コンポジションに登録されているエレメントとその状態を調べることにより、インテグレーション作業に必要なファイルを正しく特定することが容易にできる。

5.2 コンポジション型

LifeLineでは、コンポジションに対して型を定義することができる。そこで、コンポジションを作成する際に、インテグレーション作業により生成されるターゲットファイルの種類によりコンポジションの型を選択するようにしておけば、コンポジションの型を調べるだけでターゲットファイルの生成方法を特定することができる。

5.3 ターゲット種類

LifeStudio/Cでは、各フォルダ毎に生成されたターゲットファイルを格納するディレクトリを用意している。格納ディレクトリは、開発対象となるマシン機種



(a) リザーブしていない状態　　(b) d.c をリザーブした状態

図2 参照するファイル

に対して一意に決るようになっており、作業者はインテグレーション作業を行う前に、どのマシン機種に対するターゲットを生成するかを指定するだけでターゲットファイルの格納場所を特定できる。このターゲット種類はプロジェクト毎に設定されるようになっている。

また、設定されているターゲット種類を調べることにより、マシン機種によって異なるターゲットファイルの生成方法も特定することができる。

6 インプリメンテーション

LifeStudio/C はサブツールとして、コンポジション構造やフォルダ構造のブラウジングとオブジェクトの作成を支援するファイルマネージャ (FM)、ファイルとコンポジションの版管理を支援するバージョンマネージャ (VM)、インテグレーション支援するターゲットマネージャ (TM) を用意している。ここでは、TM がどの様にしてインテグレーションのメカニズムを実現しているか述べる。

6.1 Make の利用

UNIX 上でインテグレーション作業を支援するツールとして Make がよく知られており、広く用いられている。そこで、LifeStudio/C でも、UNIX 環境との融合性などの観点から、実際のインテグレーションは Make を呼び出すことによって行なうことにした。

Make を使用するためには、作業者は Makefile というファイルを作成しておかなければならない。Makefile 中には、例えばロードモジュールを作成する場合には、リンクするために必要なオブジェクトファイル名の一覧やヘッダファイルへの依存関係といった情報を記述しておかなければならない。また、ロードモジュールを作成するのに必要な構成ファイルが追加・削除されるたびに、Makefile を修正しなければならない。この手間を軽減するために、TM は Makefile 生成の機能を提供している。

6.2 Makefile の生成

各ターゲットに依存した Makefile は、あらかじめ用意されるテンプレートファイル中に様々な定義情報を埋め込み、プリプロセッサ (cpp) で処理することによって生成される。

6.2.1 テンプレートファイルの選択

Makefile のベースとなるテンプレートファイルは、生成されるターゲットファイルの種類によって異なってくる。TM では、Makefile 生成時に対象となるコンポジションの型を調べ、コンポジション型毎に予め用意されているテンプレートファイルの中から、そのコンポジション型に対応したものを自動的に選択するようになっている。

6.2.2 各種定義情報

Makefile に埋め込まれる定義情報には以下のようなものがある。

1. プロジェクトで共通の定義
   参照するライブラリのディレクトリ名、コンパイルフラグなど。この定義は各プロジェクトで予め用意しておく。
2. コンポジションに依存した情報
   そのターゲットを作成するために必要なオブジェクトファイル名の一覧など。
   オブジェクトファイル名の一覧は、コンポジションに登録されているソースファイルの名前を変換することによって得ることができる。ファイル名の変換規則はコンポジション型に対して予め定義しておく。
   また、コンポジションによっては、他のコンポジションで生成されるライブラリなどのターゲットファイルを参照することもある。そのような他コンポジションのターゲットファイルがコンポジションに登録されている場合も、コンポジションに依存した情報として定義情報に含まれる。
3. 作業者が記述した Makefile
   作業者は、TM を用いて、生成される Makefile に含ませたい定義を記述することができる。テンプレートには予めいくつかの Make のターゲットが記述されているが、それら以外のターゲットを作成したい場合などには、作業者が直接 Makefile の一部を記述することができる。
4. 依存関係の情報
   各オブジェクトファイルがどのヘッダファイルに依存しているかに関する情報。依存関係の情報はソースファイルから自動的に抽出される。

これらの定義情報は、それぞれファイル中に保持される。

6.2.3 cpp 起動時の引数

Makefile を作成するために、上で述べた定義情報がテンプレートファイル中に埋め込まれ、cpp で処理される。cpp への引数として、テンプレートファイル名と定義情報ファイル名が与えられる。

また、マシン機種によるターゲットファイル生成方法の差異はテンプレートファイルにおいて条件コンパイル文 (# ifdef 文) を用いて切り分けることにより実現されるので、ターゲット種類も cpp への引数として渡される。

6.3 ターゲットファイルの格納

インテグレーション作業により生成されたターゲットファイルは、ターゲット種類に応じた格納ディレクトリに格納されなくてはならない。

TM では、この機能を実現するためにシンボリックリンクを用いている。Makefile が生成されると、ターゲット種類に対応した格納ディレクトリにあるターゲットファイルに対するシンボリックリンクが、Make を実行するディレクトリに張られる。そこで、Make が実行されると、自動的に格納ディレクトリに存在するターゲットファイルが更新されることになる。

7 評価

本稿で説明した LifeStudio/C のインテグレーション支援機能の有用性の実証ならびに評価をするために、LifeStudio/C 自体の開発に LifeStudio/C の TM を用いた。

LifeStudio/C はその機能により、ファイルマネージャ (FM)、ターゲットマネージャ (TM)、バージョンマネージャ (VM)、共通部分の 4 つのモジュールに分れる。

TM を用いる前は、Makefile を作成するだけでもかなりの作業を必要とし、リザーブ・デポジットの状態が変わる度に Makefile を含めたインテグレーション作業の環境を作業者が直接整えなくてはならなかった。そのため、インテグレーション作業時のミスが幾度か生じた。

ところが TM を用いると、一人の作業者がテンプレートファイルを用意しプロジェクト共通の情報を定義するだけで、実際のインテグレーション作業時には Makefile 生成の指示と Make 実行の指示を出すだけで良く、実際のインテグレーション作業は大幅に減少した。また、インテグレーション作業に伴うミスはほとんど無くなった。

このことから、TM の持つインテグレーション支援機能の有用性を実証することができたと考えられる。

8 おわりに

本稿では、成果物・設計情報管理機能 LifeLine を用いた C プログラム開発支援環境 LifeStudio/C におけるインテグレーション支援機能について述べ、その有用性を LifeStudio/C を用いてソフトウェア開発をすることにより実証した。

参考文献

[1] 岸知二, 入交晃司, 坪谷英昭. CASE 環境構築のためのファイル管理機能. 情処学会 CASE 環境シンポジウム, 1989 年 3 月.

[2] 坪谷英昭, 岸知二, 入交晃司, 鈴木美和子, 猪狩錦光. 構成・版管理ライブラリ LifeLine を用いた C プログラム開発環境. 情処学会 ソフトウェア工学研究会, 1990 年 7 月.

[3] Stuart I.Feldman. *Make - A Program for Maintaining Computer Programs.* Software – Practice and Experience, Vol.9, No.4, April, 1979.

[4] Steve Cichinski, Glenn S.Fowler. *Product Administration through Sable and Nmake.* AT&T Technical Journal, July/August 1988

[5] David B.Leblang Robert P.Chase,Jr. *Computer-Aided Software Engineering in a Distributed Workstation Environment.* ACM-SIGPLAN, April, 1984.

[6] Nestor,J.r. *Toward a Persistent Object Base.* Technical Report SEI-86-TM-8, Software Engineering Institute, July 1986.

[7] Peter H.Feiler. *Software Configuration Management Advances in Software Development Environments.* Tutorial Note, 12th ICSE, March 26-30, 1990