データベース・システム 76－5
（1990. 3. 12）

ソフト開発環境における情報管理について

岸 知二
日本電気株式会社
ソフトウェア生産技術開発本部

ソフト開発環境での成果物、設計情報管理機能LifeLineにおける情報管理の考え方を示しながら、ソフト開発環境における情報管理のあり方について考察する。成果物と設計情報とはお互いに関連し合っているため、その両者を適切に関連づけて管理する必要がある。LifeLineでは成果物中に定義された設計情報と成果物とを対応づけて管理する名前管理の機能を持っている。この機能を用いることにより、設計情報を一元管理する際のいくつかの問題点を補うことができる。

On Information Management in Software Development Environment

Tomoji KISHI
Software Engineering Development Laboratory
NEC Corporation
Igarash building, 11-5, Shibaura 2-chome,
Minato-ku, Tokyo, 108, JAPAN

Information management in a software development environment (SDE) is discussed. In SDE, relationship between artifacts and design information is important. We are developing object management system LifeLine, as a platform for SDE. In order to manage design information along with artifacts, name management facility is introduced into LifeLine. This name management facility also makes up for some drawbacks of centralized design information management.

1 はじめに

ソフト開発環境において、情報管理の機能は重要なコンポーネントである。ソフト開発における情報には、成果物、設計情報、管理情報など、様々なものが含まれる。そのため、あえて情報管理といわずオブジェクト管理と呼ぶことも多い。ソフト開発環境における情報管理の特性や、既存のファイルシステムやデータベースシステムを用いて情報管理を行なう際の問題点等についても報告されている [3]、[12]、[13]。

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 を開発している [9]。本稿ではこの LifeLine における情報管理の考え方を示しながら、ソフト開発環境の特性にてらした情報管理について考察する。ソフト開発環境での成果物の管理については、構成管理の研究結果に基づいた考え方が一般的になりつつあるが、本稿では特にこうした成果物管理と関連付けた設計情報管理のあり方を中心に議論する。

2 LifeLine の基本概念

LifeLine 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。プロジェクトは成果物や設計情報の管理空間である。すべ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。こうしたすべてのオブジェクトに対しては任意の属性を定義でき、またファイル中に定義されている情報は名前として管理される。これらの概念については、本稿中で必要に応じて説明を加えていくが、より詳しくは [9] を参照されたい。

3 成果物の管理

本章ではソフト開発環境における成果物の管理機能について述べる。ソフト開発は成果物を作成するための作業であり、ソフト開発環境においては、これらの成果物を管理する機能が不可欠である。成果物の中には、最終成果物としてのソースコード、ロードモジュール、各種ドキュメントなどだけでなく、中間成果物としてのドキュメント、オブジェクトコード、テストデータ等々も含まれる。成果物の管理については、構成管理 (configuration management) として各種の研究がなされてきた [4]、[5]。多くのソフト開発環境では、こうした構成管理機能を提供している [1]、[2]、[11]。

3.1 成果物の種類

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、等々がこれに含まれる。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々はこれに含まれる。LifeLine では前者をファイル、後者をターゲットという呼ぶ。

ターゲットはそれを生成するために必要な成果物としてのファイルもしくは他のターゲットと、それらのファイルやターゲットからどのようにして生成するかという生成手続きによって定義される。一般にターゲットとその生成元となる成果物は常に整合性のとれた状態になっていることが望ましい。例えばソースコードが修正された時には、対応するオブジェクトコードも再生成される必要がある。しかしソフトの開発途中では不完全な状態が生じるのが普通であり、常時整合性のある状態に保つことは困難である。従って多くの場合作業者が陽に指示をして始めて整合性を保持するメカニズムが働くようになっている。こうした機能としては MAKE[7] がよく知られているが、LifeLine においても作業者がターゲットの生成を陽に指定することにより、必要な生成手続きが起動され、ターゲットを生成するようになっている。

3.2 構成情報の管理

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 [12]。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。これはファイル、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。前述したターゲットの定義にはその生成に必要となる成果物の集合を指定する必要があるが、その指定にはコンポジションを用いることができる。構成情報はソフト開発の進行に伴って変化していくが、その構成に対応するコンポジションを修正さえすれば、ターゲットの定義にもそのまま反映される。

3.3 版管理

成果物の版管理機能としては SCCS[14] や RCS[15] が広く用いられているが、

ソフト開発環境でも、個々のファイルの版管理には同様の差分管理の技法が用いられることが多い。一般に版管理の対象となるのはソースコード等のファイルであり、オブジェクトコードやロードモジュールは対象とならない。構成管理においてはこうしたファイル単体の版

管理だけでなく、構成情報そのものの版管理をする必要がある。LifeLine はコンポジションに対する版管理機能を提供している。従ってファイルに対しては SCCS や RCS と同様な差分管理を適用するが、ターゲットに対してはそれに対応付けられたコンポジションとその生成手続きの記述に対して版管理を行なうことにより版を管理する。なお版の管理モデルとしては、従来的なリビジョンとバリアントによる木構造モデル [14]、[15]、[6] が用いられることが多い。

構成情報に対する版管理で留意すべき点は、様々な単位での版管理が共存するという点である。例えば一つの成果物を作成している過程ではファイル単体での版管理を行なわれ、モジュールテストを行なっている状態ではモジュールという構成が版管理され、全体が結合された時点ではシステム全体の構成が版管理される。一般にある特定の版管理の単位について過去の構成情報を取り出した場合には、システム全体としては必ずしも整合性のとれた状態にはならない。例えばリンク単位中の一つのソースファイルだけについて過去の版を取り出しても、そのリンク単位全体で整合性がとれる保障はない。こうした不整合な状態が常時起こり得ることが、ソフト開発環境の特性の一つと言える。

3.4 アクセス制御

ソフト開発環境は、複数人でのソフト開発に伴う様々な問題 (programming in the many) を軽減する役割を果たさなければならない。そのための一つの機能として、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挙げられる。構成管理ではベースラインとワークスペースでの情報のやりとりによって解決することが一般的である。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。本稿ではこの操作をリザーブ・デポジット [10] と呼ぶ。

LifeLine ではこうしたベースラインやワークスペースを表現するために、プロジェクトを用いる。プロジェクトにはベースラインに相当するルートプロジェクトと、ワークスペースに相当するサブプロジェクトの二種類がある。開発作業に対しては一つのルートプロジェクトと、その子供として定義される複数のサブプロジェクトが用意される。一般にサブプロジェクトは作業者毎に作成される。サブプロジェクトがさらにサブプロジェクトを持つこともある。サブプロジェクト中で成果物等の修正作業等を行なう時には、リザーブ・デポジットの操作を用いて行なう。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。

ソフト開発環境においては、修正作業等に人手の作業が介在するため、ベースラインとのやりとりの時間的スケールは一般に大きい。場合によっては何日、何週間といった単位でリザーブし続けることもあり得る。そのためリザーブ中であっても他の作業者が参照だけはできるようしておく必要がある。NSE ではワークスペースへのコピーは全く作業者毎に自由に行なわせ、ベースラインに返す時点で他の作業者が修正が修正を行なっていないかどうかをチェックし、修正が衝突した場合にはマニュアルでその修正をマージする手法をとっている [1]。すべての状況でマージが必ずうまくいくわけではなく、楽観的なアプローチではあるが、一つの考え方であるといえる。LifeLine ではリザーブ・デポジットによる厳密なアクセス制御を採用しているが、版リザーブ・版デポジットの機能を用意することにより、特定の成果物であっても、異なるバリアントに対しては同時に修正作業が可能なようにしている。

3.5 作業者のビュー

開発作業時には、作業者毎に異なったビューで情報を見る必要がある。例えば一人の作業者がソースコードを修正している場合、その作業者は修正途中のソースコードを見ながら作業をしなければならないが、他の作業者はその修正が完了するまでは、修正途中のソースコードを見ることは望ましくない。また修正が終了してデポジットを完了した時点では、二人の作業者は同一のソースコードを参照しなければならない。LifeLine ではリザーブ・デポジットの状況を見て、プロジェクト毎のビューを一意に決定している (図1)。

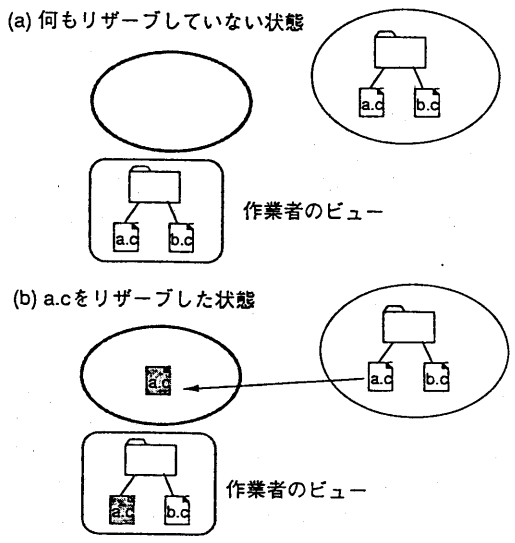


図1 作業者のビュー

また同一の作業者であっても、作業の内容に応じて必要となる成果物の集合は異なる。こうした作業内容に応じたビューは、コンポジションを指定することによって決定される。例えばリンク単位毎にコンポジションによって構成を表現し、作業対象となるリンク単位を含むコンポジションを指示することによって、その作業に必要な成果物の集合を、必要な構成で参照することができる。

4 設計情報の管理に関する考察

本章では設計情報の管理に関する考察を行なう。

4.1 成果物中に定義される設計情報

構成管理機能をベースとした成果物管理では成果物が管理の基本単位となっており、それに対しては何のセマンティクスも与えられない場合が多い[8]。しかし実際には成果物中には様々な設計情報が含まれている。ソフト開発作業は、様々な成果物を作成していく過程として捉えることもでき、すべての設計情報は成果物中に何らかの形で記述されていると考えることができる。従ってソフト開発環境においては、成果物中に定義された設計情報を管理することができなければならない。

こうした設計情報は成果物と別々に管理されるべきものではない。例えばオブジェクトコードを再生成する必要があるかどうかを判定する際には、単にそれに対応するソースコードとオブジェクトコードが正しく対応しているかをチェックするだけでは不十分であり、ソースファイルがインクルードしているファイルとの対応関係も調べる必要がある。しかし一般にソースファイルが何をインクルードしているかは、そのソースファイル中の文字列を見ないと判定できない。すなわち前節で述べたターゲットの生成を正しく行なうには、ソースファイルという成果物中に定義されたインクルード関係という設計情報を参照しなければならない。またシステム構成やモジュール構造を記述した図式は、それ以降に作成される成果物の管理構造等と深い関連を持ち得る。このように成果物管理と成果物中に定義されている設計情報の管理は密接な関係があり、適切に関連付けて管理される必要がある。

4.2 設計情報の一元管理

設計情報を一元管理することによるメリットはいろいろと考えられる。例えば開発を通じて一つの用語辞書を参照することにより、全体として用語を統一することができる。また一度入力された設計情報を重複して入力する無駄を省いたり、一箇所で設計情報の修正を行なうことにより、その設計情報を用いているすべての箇所で同様の修正を自動的に行なうことも可能となってくる。

しかし反面、設計情報の一元管理には以下のような問題点もある。

- 一般に一元管理は設計情報関の整合性を強く要求するが、開発環境中ですべての設計情報の整合性を厳密に要求することは、実際的ではない。ソフト開発作業は最終的に整合性のある複数の成果物を作り上げていくプロセスであり、その過程ではシンタクス的、セマンテクス的に不完全な成果物が多く作られ、管理される局面が必ずある。そのような過程のあらゆる時点で常に整合性を保つことは現実問題として困難である。

- 間違った情報を入力した時のリスクが大きい。複数の成果物に関連する設計情報を間違って修正した場合にはその操作によって正しい情報がなくなってしまうことがありうる。またモジュール構造の様な設計情報は、他の成果物の管理構造を規定するため、そうした設計情報の修正のもたらす影響は極めて大きい。

4.3 情報の粒度

ソフト開発環境では一つの設計情報が様々な粒度(granularity)で扱われることも多い。例えば関数という設計情報は、モジュール構成図中ではそれ以下の構造を持たないプリミティブな情報として出現するするが、ソースコード中では、さらに下位の構造を持つ情報として扱われるかもしれない。従って情報の粒度は、それを扱う成果物の種類等によって、相対的に決定されなければならない。

ソフト開発環境で用いられるツールやそれらのツールの使われ方を始めから固定することができるならば、それぞれの設計情報の粒度をあらかじめ固定して考えることも可能であるが、ソフト開発環境では多くの場合これを固定することは困難である。特に近年は開発環境がオープンであることが強く求められており、こうした設計情報の粒度を柔軟に決定できることが要求されている。

5 階層的な情報管理

前章での考察を踏まえ、LifeLine では成果物と対応づけた設計情報の管理のために名前管理と呼ばれる機能を持っている。これは成果物中に定義された設計情報を、その成果物に適切な粒度で管理する機能である。この機能は用語辞書のような一元的な設計情報管理機能に置き代わるものではなく、むしろ補完的に使われるものであると考えている。従ってLifeLine による情報管理は、図

2 に示すような階層を持つことになる。

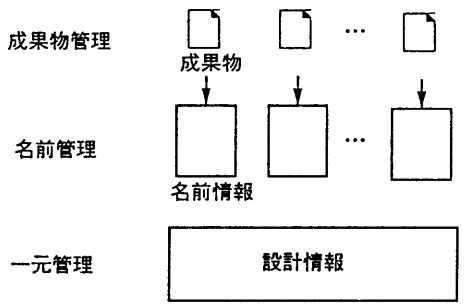


図2 階層的な情報管理

5.1 名前管理

名前は成果物に対応付けて管理され、対応付けられた成果物中に定義されている設計情報をその成果物に適した形態で管理したものである。例えばソースコードに対しては、その中にどのような関数や変数が定義されているか、どのようなコール関係やインクルード関係があるか、等が管理される。こうした名前は単項関係や二項関係といいう形態で格納される。これらの関係は様々な属性を持つことができる。定義されている関数の型やパラメータという情報は属性として表現される。また定義されているファイル中での行数などを属性として持つことにより、名前から対応する成果物中の位置を特定することができる。

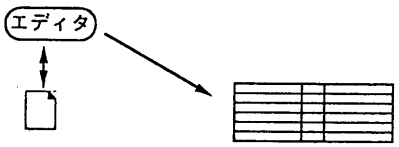
名前は成果物と対応付けられているため、あくまで「この成果物中にはこのような設計情報が定義されている。」ということを示すだけである。従って名前にアクセスする際には、その成果物をスコープとして指定する。例えば「*a.c* というソースコード中に定義されている関数 *f* の型は何か？」とか、「*b.doc* という関数仕様書中に定義されている関数 *f* の型は何か？」といった形でアクセスされる。こうしたスコープの決定にはコンポジションを用いることができる。例えばコンポジションを用いてソースコードのリンク単位を表現するならば、リンク単位中に定義されている関数を検索する、といったことが可能となる。またあるファイル中で「関数 *f* が関数 *g* をコールしている。」という定義がされていても、その *g* がどのファイル中に定義されている関数なのかは一般に判定できない。こうした関係の検索においても、コンポジション等を用いて検索のスコープを指定することにより、始めて実際の対応関係が決定される。

なおソースコード中の関数の型と関数仕様書中の関数の型とが本来同一でなければならない、などといった意味付けは、名前管理自身では行なわず、その外側でなされる。一般に成果物中の設計情報に対しては様々な意味付けが求められるため、それを名前管理で規定することは難しい。例えば一つの設計情報の修正を他に波及させたい場合もあるし、単に異なることを検出したいだけの場合もある。また一般に二つの成果物中で定義される設計情報が異なる場合、どちらが正しいかという判定は不可能である。従ってこうした意味づけはソフト開発環境側に任さざるを得ない。

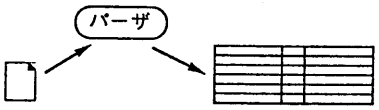
5.2 名前の登録

名前を登録する手段としては、以下のような方法がある (図3) 。

- 成果物をファイルにセーブする時に、同時に名前を登録する。この場合、成果物を作成するためのツール (エディタ等) が、成果物中の設計情報を取り出して、名前として登録する機能を持っている必要がある。
- セーブされたファイルから設計情報を取り出して名前管理機能に登録する。この場合は、成果物を作成するためのツールが名前を登録する機能を持つ必要はないが、ファイルから設計情報を取り出すためのパーザが必要となる。



(a)成果物作成時に登録



(b)必要に応じてパーザが起動され登録

図3 名前の登録

パーザを用いる場合には、登録した後に成果物が修正されると、再度パーザをかけて再登録をしなければならない。これはターゲットの整合性の保持と同様である。LifeLine では、前述したターゲットの機能を拡張して、設計情報にアクセスした時に、その設計情報をとり出した時間と成果物の生成時間を比較して、必要であればパーザを自動的に起動して再登録することができるようになっている。そのため常に成果物の内容と一致した設計情報を得ることが可能である。

5.3 アクセス制御と版管理

以上述べたように、名前はその時点で成果物中に定義されている情報を抽出したものであり、成果物と一対一に対応している。すなわち名前情報の修正は成果物の修正によってしか行なうことはできないし、その参照権は成果物への参照権と一致している。原則的に名前情報そのものの版管理を行なうことはなく、成果物への版管理を行なって必要な時点で名前情報を取り出すという立場をとっている。(但し効率上の理由で、過去の版の名前情報を保存することは可能である。)

3.3で述べたように、版管理は様々な構成単位に対してなされる。コール関係のように関係として表現される設計情報は、その主語と目的語が異なる版管理単位に含まれることがあり、それぞれが別々に版管理された場合の意味づけは自明ではない。前述したように名前管理における関係の表現は、それぞれの成果物で完結している。すなわちファイル $a.c$ 中で関数 f が関数 g をコールしている場合、その情報は $a.c$ の名前情報として格納されるだけであり、その目的語である g がどの名前に対応するかは検索時のコンテキストによって判断されるだけである。このように名前管理は極めて疎な形で関係情報を保持しているため、版管理を自然に行なうことができる。

5.4 設計情報の一元管理

名前情報とは別に、従来からの用語辞書的な使い方ができる設計情報管理機能も必要となる。名前情報から必要な情報を取り出して一元管理することもあるだろうし、作業者が直接それを登録することもありうる。いずれにせよ一元管理される設計情報は開発環境全体から参照されるものであり、何らかのオーソライズのプロセスを経た情報が登録されるように運用されなければならない。この一元管理の機能と名前管理機能とはお互いに補完的な役割を持っており、ソフト開発環境においてはその両者が必要となると考えている。

6 おわりに

以上LifeLineの機能を紹介しながら、ソフト開発環境における情報管理について考察した。名前管理の機能は既に複数のソフト開発環境に適用中である。今後この経験を踏まえて機能面、性能面での一層の改善を検討していきたい。

参考文献

[1] Evan W.Adams, Masahiro Honda, Terrence C.Miller, *Object Management in a CASE Environment.* Proc. of the 11th ICSE, May, 1989.

[2] N. Belkhatir, J.Estublier, *Experience with a Data Base of Programus.* Proc. of the ACM Sigsoft/Siglpan symposium on practical software development environments, December 1986, SIGPLAN Notices, vol22, No.1, 1987.

[3] Philip A.Bernstein, *Database System Support for Software Engineering – An Extended Abstract –.* Proc. of the 9th ICSE, March, 1987.

[4] Wayne A.Babich, *Software Configuration Management.* Addison-Wesley, 1986.

[5] Edward H.Bersoff,Vilas D.Henderson, Stanley G.Siegel, *Software Configuration Management.* Prentice-Hall, INC, 1980.

[6] Jacky Estublier, Jean-Marie Favre, *Structuring Large Versioned Software Products.* IEEE, Proc of COMPSAC 89, September, 1989.

[7] Feldnam S.I., *Make - A Program for Maintaining Computer Progrrams.* Software – Practice and Experience. Vol.9, No.4, April, 1979.

[8] T.Gallo, G.Serrano, F.Tisato, *ObNet: an object-oriented approach for supporting large, long-lived, highly configurable systems.* Proc. on the 11th ICSE, May, 1989.

[9] 岸知二, 入交晃一, 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.

[10] Charles W.Krueger, *The SMILE Reference Manul.* The GANDALF System Reference Manuals. Carnegie Mellon University, 1986.

[11] David B.Leblang, Robert P.Chase, Jr., *Computer-Aided Software Engineering in a Distributed Workstation Environment.* Proc. of the ACM Sigsoft/Siglpan symposium on practical software development environments, April 1984.

[12] John Nestor, *Toward a Persistent Object Base.* Techinical Report SEI-86-TM-8, SEI, July 1986.

[13] John Nestor, Richard Snodgrass, *Database Technology for Software Engineering.* Tutorial Notes, the 9th ICSE, March, 1987.

[14] Marc.J.Rockhind, *The Source Code Control System.* IEEE Trans. on Software Engineering, SE-1(4), December, 1975.

[15] Walter F.Tichy, *RCS: A Revision Control System.* Integrated Interactive Computing Systems, North-Holland Publishing Company, 1983.

[16] Walter F.Tichy, *Tools for Software Configuration Management.* International Workshop on Software Version and Configuration Control, Grassau FRG. January 1988. Published in Springler Verlag.