プログラミング言語 21 － 3
（１９８９．６．１３）

プログラミング言語のための ソフトウェアデータベースについて

石原博史　　徳田雄洋

東京工業大学工学部情報工学科

概要

ソフトウェア開発の大規模化にともない、開発時に生成される様々なデータの保守、管理を行うソフトウェアデータベースが要求される。本論文では、ソフトウェアデータベースに必要な諸機能を述べ、この機能を実現するためのアプローチとして、ファイルシステム、ハイパーメディア、従来のデータベースシステム、オブジェクトベースの４つの現状を検討する。最後にこれらのアプローチによるソースコードの表現方法を比較することで個々の問題点を述べる。

Software Databases for Programming Languages

Hiroshi Ishihara and Takehiro Tokuda

Department of Computer Science, Tokyo Institute of Technology

Abstract

With the increase of the size of software development projects, we need software databases which manage various type of data created in the development process . In this paper, we discuss necessary functions for software databases, and we examine four approaches to the construction of software databases: file systems, hypermedia, conventional databases, and objectbases . Finally, we compare representation method of the source code using these approaches, and summarize related technical problems .

1 はじめに

ハードウェアの性能が向上していくにつれて、ソフトウェアに対する要求が大規模な、かつ高度なものになりつつある。しかし、それに比例してソフトウェアの生産性が向上したとはいえない。いかにしてソフトウェアを効率よく生産するかという問題は、ソフトウェアが大規模になればなるほど重要な問題として現れてくる。近年、この問題を解決しようとする様々な研究が行われている。ソフトウェアの設計プロセスの形式的な記述に関する研究、ソフトウェア開発環境に関する研究などがそれにあたるが、そのなかに、ソフトウェアデータベースに関する研究がある。ソフトウェアデータベースとは、ソフトウェアの開発時に生成される全てのデータを保守、管理するものである。生成されるデータとしては、ソースコード、オブジェクトコードはもちろんのこと、要求仕様、設計書などの文書、ユーザ用、管理者用マニュアル、開発スケジュール、入力データ、出力データなど様々である。これらのデータの間には複雑な相互関係があるが、ソフトウェアデータベースはこの関係についても管理を行う。開発者はこれらのデータを登録、修正、削除することで目的のソフトウェアを作成するのだが、データの変更によって他のデータに矛盾が生じないようにすることがソフトウェアデータベースに求められる。

従来の商用データベースでは、定型のデータを大量に効率よく管理し、高速に検索する機能が注目されているが、ソフトウェアデータベースではこれと異なる機能が要求される。例えば、多種多様なデータを表現するためのデータモデル、バージョン管理、排他制御、照会などが必要な機能である。現在、これらの機能をすべて満たす実用的なものはあまり見られないが、ソフトウェアデータベースを実現するためのアプローチはいくつか考えられる。

本論文では、ソフトウェアデータベースに必要な諸機能を述べ、この機能を実現するソフトウェアデータベースへのアプローチとして、ファイルシステム、ハイパーメディア、従来のデータベースシステム、オブジェクトベースの4つを考察し、最後にこれらを比較・検討することによって将来のソフトウェアデータベースのあるべき姿を検討していきたいと思う。

2 ソフトウェアデータベースに必要な機能

ソフトウェアデータベースで扱うデータは多種にわたり、データ間の関係も複雑である。そのため、定型データを大量に扱うことを目的とする商用データベースとは全く異なる、ソフトウェア開発に特有の機能が必要である。以下にソフトウェアデータベースに必要な主な機能を述べる。

2.1 バージョン管理

一般にソフトウェアは機械部品や、建物のようなハードを作るのとは違い、製作過程の履歴を残しておくことで簡単に過去の状態へ戻すことができる。ソースファイルやプログラムへの入力データなどは、誤って消去してしまうと痛手が大きい。また修正変更を行なう場合も注意が必要である。例えば、バグの対処でソースファイルを修正した場合に、より重大なバグを埋め込んでしまうことがある。このような場合にそのバグが発見された時点でなぜこのような修正を行なったのかなどがわかれば再修正にかかる手間も省けるであろうし、対処しきれない場合は以前のバージョンを作り直すことも必要になるであろう。このような理由でソフトウェアの開発にはバージョン管理の機能 [7] が必須である。

2.2 データ間の関係管理

ソフトウェアデータベース上のデータ間には様々な関係がある。例えば以下のものが考えられる。

- 定義と使用の関係

- オブジェクトコードとソースコードの依存関係
- 複数のオブジェクトから1つのモジュールを作成するバージョン関係
- 複数のモジュールから1つの製品を作成するコンフィグレーション関係

これらの関係をシステムの中に直接的に表現し、管理もシステムが行うことが要求される。この関係はソフトウェア開発上で動的に変化していくが、この変化によってデータ間に矛盾が生じないように監視しなければならない。関係を管理することで、実行形式の作成の自動化や、ソースコードに対する照会機能が実現できる。

2.3 排他制御

1つのトランザクションにかかる時間が数秒のように短いものであれば同じデータへ複数人がアクセスすることは希なので、データに対して強力な排他制御を行えばよい。しかし、ソースコードの修正のように1回のトランザクションが数分から数時間と極端に長い場合、データへのアクセスが衝突する機会は無視できず、強力な排他制御を行ってしまうと、修正が終わるまで、他のユーザは待たなければならなくなる。そのため、比較的楽観的な排他制御が必要になる。

2.4 システム異常からの回復処理

従来のデータベースの処理は、レコードを自分のプロセス内に読み込み、レコードになんらかの操作を行い、修正されたレコードをまたデータベースに戻すことの繰り返しである。処理時間は普通は数秒、多くても数分である。システムの異常によって処理が失敗したとしてもユーザがその処理を再度実行すれば済む程度のものである。しかし、ソフトウェアを扱う場合、その処理時間は普通で数時間、多いときは数日にわたり、システム異常による被害は重大なものになることが多い。そのため、システム異常からの回復機能、または、被害を最小限に抑えるための機能が必要である。

3 ソフトウェアデータベースへのアプローチ

ソフトウェアの開発の過程では、プログラムソース、オブジェクトコード、要求仕様、設計書などの文書、ユーザ用、管理者用マニュアル、開発スケジュール、入力データ、出力データなどのような様々なデータが生成される [1]。従来の開発環境では、これらはそれぞれ別々に管理されている場合が多かった。管理する方法もさまざまで、それぞれを計算機のファイルとして、あるいは文書（物理的な紙）として、また人の経験や記憶として保存されてきた。単独の、または数人で行うような小規模な、時間的にも短期間なソフトウェア開発ならばこの方法でも効率はどうであれ可能であった。しかしソフトウェア開発が大規模になっている現在では、これらを別々に管理することは混乱の原因になりかねない。

これらのデータや、データ間の関係を効率的に処理するためには効率の良いデータの表現方法を考えなければならない。このデータ表現でソフトウェアデータベースへのアプローチを分類すると以下の4つが考えられる。

- 従来のファイルシステム
- ハイパーメディア
- 従来のデータベースシステム
- オブジェクトベース

明示的にソフトウェアデータベースとは表現されてはいないが、これらの方法を利用した具体的なシステムも存在するので、それらも交えて以下に詳しく述べていく。

4 ファイルシステムによるアプローチ

ソフトウェア作成過程で生成されるデータをファイル単位で表現し、一連の処理も直接そのファイルを操作することにより行う方法である。

4.1 生のファイルシステム

データを直接ファイルシステム上のファイルとして格納する方法である。ソフトウェアをファイルシステム上で開発する場合、あるディレクトリにソースコード、オブジェクトコード、ちょっとした文書やメモなどを作成し、そこで作業を行うのが普通であり、小規模のソフトウェア開発における一般的な方法である。これも1つのデータベースと考えられるが、ファイルシステム自体はこのファイルがソースファイルなのか、文書なのかは知らないし、また、ファイル間の関係を管理するのは、ファイルシステムではなくユーザが行わなくてはならない。そのため、ファイルの内容に関連した知的作業を行うことができないのである。

ソフトウェアデータベースに要求される機能も、せいぜいファイルのオーナがそのファイルのパーミッションを操作することによって排他制御を行う程度であり、ファイルシステム自身のみでは、ソフトウェアデータベースを実現するのは難しい。

4.2 RCS と SCCS

現在UNIXの上には、有名なrevision control system として SCCS[10] と RCS[13] がある。SCCS はベル研で作られたシステムで、System V で用いられている。RCS はパーデュー大学で作られたシステムで、4BSD で用いられている。Revision control システムとは数ある revision を一括して管理しようとするものである。

Revision control システムの基本的な操作は、管理ファイルの作成、特定バージョンの回復、バージョンの登録の三つである。この操作の作業の流れとしては1のようになる。

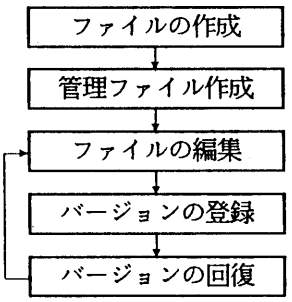

図 1: revision control システムの作業の流れ

この作業の流れは一般のデータベースの変更の処理に大変似ている。その他、ファイルのlock と unlock 、複数バージョンの join などのようなデータベースの基本的機能も備えている。

このように、バージョン管理、排他制御の機能を備えているが、ファイルシステムがファイルの内容を認識していないと同様に、RCS や SCCS も単なるテキストファイルを行単位で管理しているだけなので、何を管理しているかはシステムは知らない。そのため検索や、照会などの知的処理を行うことはできない。

4.3 Gandalf

Gandalf は大きなソフトウェア開発プロジェクトをサポートするデータベースシステムである。現在の Gandalf システムはソフトウェア開発というよりは、構造エディタの開発をサポートするものになっているが、ファイルシステムをベースとした強力なデータベースシステムである。Gandalf のデータベースはファイルシステムの木構造を利用して表現されている。ソースコードはその言語の構造に基づいて分解されてデータベースに格納される。つまり、ソースコードはデータベースの部分木で表現することができる。

5 ハイパーメディアによるアプローチ

ハイパーメディアを明確な定義するのは難しいが[8]、だいたい以下のような性質を持つものの総称である。

- 情報は小さなユニットにまとめられ、各ユニットはテキスト、ビットイメージ、音声、アニメーションなどの様々な形態を含む。
- ユニット同士はリンクによって結合されている。ユーザはリンクをたどることでユニットからユニットへ移動することができる。
- ユニットを作成、編集、リンクすることによって、ユーザは様々な目的の構造を構築することができる。

以上のようなハイパーメディアの概念を用いたものに電子辞書や、電子百科辞典などがある。辞書や百科辞典は、物理的には項目の集合体だが、理論的には複雑に絡み合ったクロスリファレンスの集合である。ユーザはある項目を参照し、その説明の中でわからない単語や項目があればさらにそれを参照するといった調子で、ランダムに検索を行う。しかも、絵や音声も項目の中に混在させることもできるのがハイパーメディアの特徴の1つである。

電子辞書のような、クロスリファレンスの中を自由にブラウジングする機能は、ソフトウェアデータベースを実現するためには必要な機能の1つであり、ハイパーメディアのリンクの概念からそれを実現することは簡単である。しかし、ハイパーメディアでソフトウェアデータベースを実現するにはいくつかの問題点がある。

1つは、ハイパーメディアで用意されているノードやリンクなどの単純なデータモデルをどのようにソースコードのような複雑な構造を持つものに適応するかが問題である。ソースコードの言語に沿った処理を行うには、システムはソースコードを扱っているということを認識する必要がある。つまり、システムが言語の構造を理解し、ユニットやノードによってソースコードを表現するための適当なデータモデルをユーザが用意しなければならない。ところで、関数型や、手続き型言語を見ると、ソースコードの内容は、定義と、その定義の使用の集合とみることもできる。システムが言語の構造を理解していれば、この定義と使用の関係を、そのままノードとリンクの関係に適応することができ、ソースコード内のブラウジングが可能になる。

もう1つは、ハイパーメディアは頻繁に変更される情報に弱い。これは辞書や百科辞典などのような、データモデルが静的なものを仮定しているからである。システム自身が情報の内容を編集し、操作することはなく、ユーザが明示的に編集しない限り情報は変更されないのである。

6 従来のデータベースシステムによるアプローチ

従来のデータベースの分野では、データモデルについては階層、ネットワーク、リレーショナルと変化していき、それと共にデータ処理や、検索、照会の技術も進歩していった。しかし、従来のデータベースは比較的簡単な定型のデータを大量に扱うことに優れており、ソフトウェアプロセスの生成物のように、可変長で、様々なデータタイプを持つものを扱うには適さない。

しかし、ソフトウェアプロセスの生成物を直接扱うのではなく、これを定型のデータに変換して扱うシステム、つまり、ユーザは従来のデータベースであることを意識せずに処理を行えるシステムが、数は少ないが、いくつか存在する。これらは、データベースの研究分野からではなく、ソフトウェア開発環境の研究分野に多くみられる。

例としてSLCSE（The Software Life Cycle Support Environment）[12]を簡単に紹介する。このシステムにはユーザ・インターフェイス、ツール、データベースの3つの概念がある。ユーザはユーザ・インターフェイスを介して操

作を行い、このユーザ・インターフェイス自身がツールを扱う、そしてツールがデータベースを操作するようになっている。

SLCSE のデータベース部の概略を 2に示す。

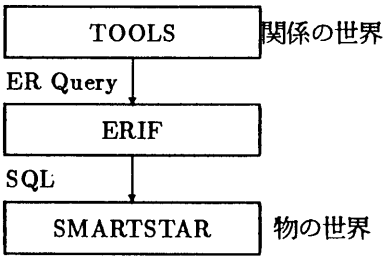

図 2: SLCSE Detabase

SLCSE では核となるデータベースに SMART STAR という商用の関係データベースを使用している。この関係の世界と、物の世界を仲介する役割をするのが ERIF（Entity-Releationship InterFace）である。ツールはこの ERIF にデータベースへの処理を命令し、その命令を SQL（Structured Query Language）に変換し、SMARTSTAR が実際の処理を行う。

ERIF で使われているデータモデルは、Entity-Relationship モデルである。これは、実世界の物と、物の関係を抽象化したもので、モデル化されるオブジェクトを表す"Entity"、オブジェクトの特性を表す"属性"、Entity 間の関連を表す"関係"の 3つの概念によって構成される [6]。このデータモデルは専門知識を持たない人でも容易に理解することができ、その豊かな表現力と、使いやすさから、工業分野と研究分野から注目されているデータモデルである。

このように、従来のデータベースを利用してソフトウェアデータベースを実現しようとすると、SLCSE の ERIF のようなインターフェイスが必要となってしまう。

7 オブジェクトベースによるアプローチ

従来のデータベースで保守、管理されるものは顧客名や、在庫数などのようなまさしく"データ"であった。しかし、ソフトウェアのように"物"を保守、管理するデータベースはオブジェクトベースと呼ぶことが多い。この点ではソフトウェアデータベースはオブジェクトベースの 1つである。オブジェクトベースで扱うデータはオブジェクトと呼ばれ、データモデルの基本単位を表すものである。近年のデータベースや、データマネージメントの分野でも、

「はじめに物ありき」という概念を利用したオブジェクト指向のデータベースの研究が盛んである。最近のオブジェクト指向のデータベースのアプローチを大別すると次のようになる [4]。

- Smalltalk のメッセージパッシングを利用したシステム [3, 9]
- サイズが変化するオブジェクトをサポートするように拡張されたシステム [2]
- 関係データモデルを変えずにオブジェクト指向的要素を支援するもの [11]

例えば、[9] では GemStone というデータベースシステムを開発している。これは Smalltalk システムを直接利用している。Smalltalk は強力なユーザインターフェイスとアプリケーション開発のためのたくさんのツールを用意しているが、シングルユーザ、メモリベース、シングルプロセッサのシステムであるためデータベースシステムに必要な機能を満たしていないので、これらの機能を追加して実現している。

ソフトウェアデータベースでは、静的なオブジェクトタイプのほかに、仕様の変更や、開発ツールの操作方法の変更などにより、動的に変化することのできるオブジェクトタイプが必要である。つまり、データベースの構造をその処理の中で動的に変化させ、拡張できる機能をオブジェクト指向のデータベースに要求している。

さらに、永続的なオブジェクトと一時的なオブジェクトの表現方法、処理方法も要求される。

8 比較・検討

以上のアプローチを比較するのに適した評価方法は現在のところあまり見られない。ここでは具体的な比較を行うために、共通例題としてソースコード、特にC言語のソースコードを4つのアプローチに基づいて表現する方法を考えてみる。

8.1 ファイルシステム

ファイルシステムの構成単位はいうまでもなくファイルである。Cのソースコードであろうが、文書であろうがデータはファイルで表現される。RCSやSCCSも同様である。Gandalfでは、プログラムは言語の構造に基づいて分解され、ファイルシステムの木構造を利用して表現される（3)。例えば、1つの関数や、グローバル変数の宣言が1つのテキストファイルに相当する。

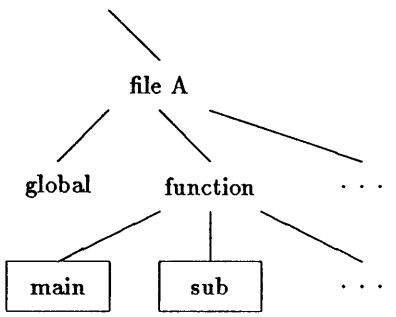

図3: Gandalfのデータ表現

8.2 ハイパーメディア

1ユニットでソースコードのどの部分を表現するかは、データモデルの設計者により決定されるが、リンクを利用した階層構造で表現する方法が考えられる。システムが定義と使用の関係を管理し、さらにシステムが自動的にこの関係にリンクを張れば、強力なブラウジング機能を提供することになる。しかし、ハイパーメディアでは基本的にデータを変更できるのはユーザに限定されており、システム自身がリンクを張るような動的なデータ修正の機能を備える必要がある。

8.3 従来のデータベース

従来のデータベース、特にここでは関係データベースを考える。関係データモデルでは、基本的にはデータベースを2項関係の集合として捉える。Cに限らず、関数型、手続き型言語の構造は階層的に捉える方が自然であり、この階層構造を2項関係に効率的に変換する方法が要求される。さらに、関係データモデルではデータの順序関係は存在しないので、Cのマクロ定義などのソースコードの意味が出現順序に依存する場合、明示的に順序関係をデータとしてユーザが定義しなければならない。

8.4 オブジェクトベース

1つのソースファイルをオブジェクトに分解し、羅列しただけでは言語にみられる階層構造を表現できない。そこで、複合オブジェクトの概念を用いて実現する。これは、オブジェクトの集合もまたオブジェクトという考え方で、親子関係を表すための基本的なオブジェクトである。複合オブジェクトを用いることで階層構造を自然に表現できることはいうまでもない。

しかし、複合オブジェクトを実際に実現するシステムにとっては、子オブジェクトを削除した場合の親オブジェクトや、親オブジェクトを削除された子オブジェクトを矛盾なく合理的に処理する機能を新たに要求されることになる。

9 おわりに

ソフトウェアデータベースでは従来のデータベースの機能の他に、様々な特有の機能を兼ね備える必要があることは述べた。前述した4つ

のアプローチのそれぞれに、必要な機能がいくつか見られるが、この４つの中には決定的なアプローチ方法は見つからなかった。しかし、ソフトウェアデータベースに期待と関心が集まる現在、必要な機能とさまざまな問題点をより明確にしていくことでよりよいソフトウェア開発環境の実現に近づいていくであろう。

参考文献

[1] Philip A. Bernstein, "Database System Support for Software Engineering", Proceedings of the 9th International Conference on Software Engineering , Monterey , CA , March 1987 , pp 166-178.

[2] M.J.Carey , D.J.DeWitt , J.E.Richardson , and E.J.Shekita, "Object and file management in the EXODUS extensible database system" , in Proc. Twelfth Int. Conf. Very Large Databases , Aug.1986 , pp.91-100.

[3] G.Copeland and D.Maier, "Making Smalltalk a Database System", Proceedings of the ACM SIGMOD , Boston , Mass. , June,1984.

[4] K.Dittrich and U.Dayal,Int. Workshop Object-Oriented Databases, Pacific Grove , CA , Sept.23-26,1986.

[5] Goldberg,A. , and D. Robson, "Smalltalk-80:The Language and Its Implementation", Addison-Wesley , 1983.

[6] John G.Hughes, "Database Technology,A software engineering approach", Prentice Hall International series in Computer Science.

[7] Katz,R.H.,Chang,E.,and Bhateja,R., "Version Modelling Concepts for Computer Aided Design Databases", Proc.1986 ACM SIGMOD Int.Conf. on Management of Data, Washington DC,May,1986,pp.379-386.

[8] ROBERT M. AKSCYN , DONALD L. McCRACKEN and ELISE A.YODER, "KMS:A Distributed Hypermedia System for Managin Knowledge in Organizations", Communications of the ACM , July 1988 , Vol.31 , Number 7.

[9] D.Maier , J.Stein , A.Otis and A.Purdy, "Development of an object-oriented DBMS", in Proc. Conf. Object-Oriented Programming Systems,Languages, and Applications , Sept.29-Oct.2 , 1986 , pp.472-482.

[10] Rochkind,Marc J., "The Source Code Control System", Proc. First National Conference of Software Engineering, IEEE , New York , 1975 , pp.37-43.

[11] Stonebraker,M. and Rowe,L.A., "The Design of POSTGRES", Proc.ACM SIGMOD Conf.1986 , pp.340-355.

[12] T.Strelich, "The Software Life Cycle Supprt Environment(SLCSE), A Computer Based Framework for Developing Software Systems", in Proc. ACM SIGSOFT/SIGPLAN Software Engineering Sysmposium on Practical Software Development Environments, Boston , Mass. , Nev.28-30 , 1988 , pp.35-44.

[13] Tichy,W.F,"RCS — A System for Version Control", Software Practice and Experience , 15 , 1985.