プログラミング言語 24－3
（1990. 2. 9）

E-Rモデルに基づくソフトウェアデータベースの設計と実現

石原博史　徳田雄洋

東京工業大学工学部情報工学科

概要

ソフトウェア開発プロジェクトの大規模化にともない、ソースコードや文書など開発時に生成される様々なデータの保守管理を行なうソフトウェアデータベースが要求される。しかし、ソフトウェアデータベースでは従来の定型事務処理指向のデータベースと異なる機能が要求される。

本論文では、実体と関連の記述に適したE-Rモデルに基づくソフトウェアデータベースの設計法とそれに関する諸問題を検討し、具体的にUNIX上でのアプリケーション開発のためのC言語用のソフトウェアデータベースを設計、実現した。最後に、このシステムを使用して、ソフトウェアデータベースの記述におけるE-Rモデルの有効性の評価を行なう。

Design and Implementation of a Software Database Based on the E-R model

Hiroshi Ishihara and Takehiro Tokuda

Dept. of Computer Science, Tokyo Institute of Technology

2-12-1 O-okayama Meguro Tokyo,JAPAN

Abstract

With the increase in the size of software development projects, the need for software databases which manage various types of data created in the development process increases. But, software databases need functions that differ from conventional business databases. In this paper, we discuss the design of a software database based on the E-R (Entity-Relationship) model, and we implement the software database for application development. These applications may be developed in C under UNIX. Finally, we evaluate effectiveness of the E-R model in order to describe software databases by using this system.

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、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、管理し、ソフトウェアのライフサイクル全体をサポートするものである。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、会議録、入力データ、出力データなどがある。このような様々なデータを管理するためには、定型のデータを大量に効率よく管理し、高速に検索することを目標にしている従来の商用データベースにはない機能が必要となる。例えば、トークンレベルの小さいデータから、プロジェクト全体などの大きなデータまでの幅広いデータの表現方法、ソースコードや文書などのバージョン管理、データ間の様々な関係管理、さらに、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ための一貫性管理などがソフトウェアデータベースに求められる。

現在、これらの機能をすべて満たしたソフトウェアデータベースはあまり見られない。そこで本論文では、E-R モデルに基づいてソフトウェアデータベースの設計と実現を行う。E-R モデルはデータベースのデータモデルの１つで、実際の物を表す実体(entity)と実体間の関連(relationship)の記述に適している。設計、実現を行う際、E-R モデルでの可能な範囲、不可能な範囲を明らかにし、ソフトウェアデータベースにおけるE-R モデルの有効性も明確にする。さらに、本論文は対象言語としてC 言語を取り上げる。C 言語はUNIX[1]の上での大規模なソフトウェア開発にしばしば使われている。しかし、C 言語独自の問題が多々あり、それによって開発時に混乱、矛盾が生じるなど、満足のいく環境が整ってはいないというのが現状である。これらの問題や大規模、大人数での開発における問題点などを解決するにはソフトウェアデータベースが有効であると考えるのである。なお、現在のC 言語の処理系には4.3BSD やSystem V 上のものをはじめ、大型機からパーソナル用のものまで数多く存在する。そのため、ANSI(American National Standard Institute) によって標準化が行われ、最近になってANSI C バージョンのものが見かけられるようになった。しかし、現状のUNIX 上での開発環境の向上という目標から考えて、本論文で例題として取り上げるC 言語は4.3BSD 上のC 言語を採用する。

本論文の構成は次の通りである。2 章では本論文の基本概念であるE-R モデルを概説し、3 章ではソフトウェアデータベースの概説と、それに必要な機能を述べる。4 章ではソフトウェアデータベースにE-R モデルを採用した理由と関係管理について述べる。5 章ではソフトウェアデータベースのプロトタイピングとして、ソースコード照会データベースであるSQUDER について説明する。そして最後の6 章でE-R モデルに基づくソフトウェアデータベースにおける問題点と将来について考察を加える。

2 E-Rモデル

E-R モデル[1] は1976 年にPeter Pin-Shan Chen が提唱したデータモデルの１つであり、実体(entity) と実体間の関係を表す関連(relationship) を記述するのに適したモデルである。さらに、データベース設計のためのtool として図式技法であるE-R ダイヤグラムを用いている。

2.1 E-R モデルの用語

ここでは簡単にE-R モデルでの用語を説明する。基本的な用語として、人、会社、出来事など直接認識可能な"物"である実体、実体間の関係を表す関連、実体や関連の特性を表す関数である属性、属性がマッピングされる値がある。

2.2 E-R ダイヤグラム

E-R ダイヤグラムは設計される概念スキーマを図示する技法で、実体集合を四角で、関連集合をひし形で表す。実体集合は関連集合で結ばれ、その線上

[1]UNIX は AT&T ベル研究所の登録商標である。

に1:n、m:n、1:1のマッピングが表示される。図1に生産会社を表したE-Rダイヤグラムの例を挙げる。

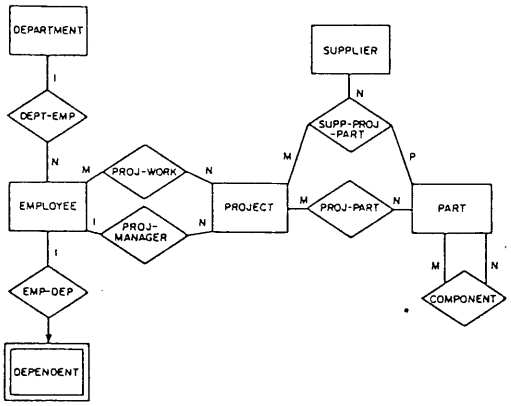

図1: 生産会社のE-Rダイヤグラム例

3 ソフトウェアデータベース

3.1 ソフトウェアデータベースの概説

ソフトウェアデータベースは、ソフトウェアの設計、開発、保守などから生成されるさまざまなデータの保守、管理を行い、ソフトウェアのライフサイクルを効果的に支援するものである(図2参照)。生成されるデータは、ソースコード、オブジェクトコードはもちろんのこと、要求仕様書、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・出力データなど現在のデータベースでは考えられなかったものなど広範囲にわたる[2]。

ソフトウェアデータベースではこれらのデータは独立に存在しているのではなく、相互に複雑な関係を持っている。この点で言えば、ソフトウェアデータベースは、データを接点とし、関係を枝とするネットワークであると考えることができる。さらにソフトウェアデータベースの特徴としては、このネットワークがソフトウェア開発中に動的に変化していくことである。データ間の関係としては、例えば以下のようなものがある。

- 定義と使用の関係
- 包括(親子)関係
- オブジェクトコードとソースコードのコンパイル関係

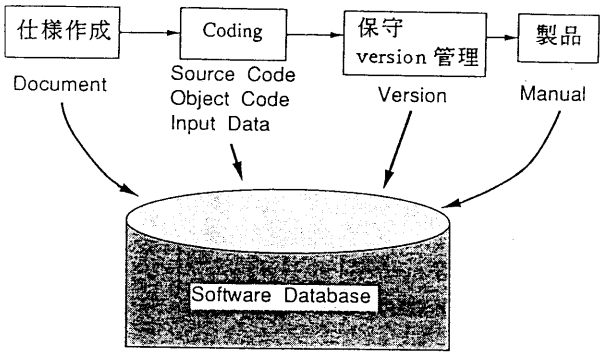

図2: ソフトウェアプロセスで生成されるデータ

- バージョン関係
- コンフィギュレーション関係

これらの関係を管理することで、実行形式の作成や、ソースコードに対する豊富な照会機能が提供できる。

3.2 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タイプもその大きさも多種にわたり、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になる。以下ではソフトウェアデータベースに必要とされる主な機能を述べる。

3.2.1 幅広い粒度に対するデータ表現

ソフトウェアデータベースで扱うデータ(オブジェクト)の大きさは、関係データベースで扱われる比較的定型のデータと違いさまざまである。小さいオブジェクトとしては開発言語のセンテンスやトークンレベル、極端に言えば1文字、1バイトもオブジェクトとして考えられる。オブジェクトの粒度レベルが小さければ小さいほどユーザからの細かな質問(query)に答えられ、そのオブジェクトの意味をシステムは細かく認識することができる。しかしそれ以上に、オブジェクトの量が莫大に増加し、1データを処理するためにその数倍のスペースが必要になり、処理時間も長くなり、オブジェクト管理の非効率化につながる。大きいオブジェクトの場合、実行形式

や、文書全体、プロジェクト全体などが考えられる。しかし、この大きいオブジェクト全体を１つのオブジェクトと考えると意味的に希薄なオブジェクト集合になりかねない。例えば、ソースファイル１つを１つのオブジェクトとみなすと、その内容に関する情報をそのオブジェクトが管理していないため内容に関する質問はできない(図3参照)。

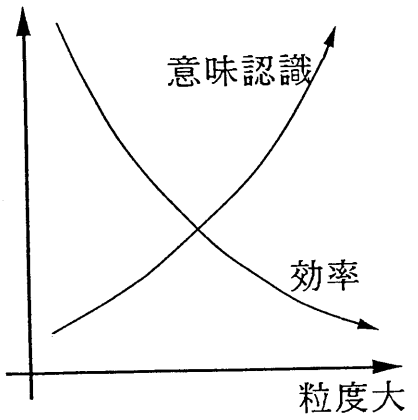

図3: データの粒度

しかも普通大きいオブジェクトはその内容が複数のオブジェクトで構成される、複合オブジェクトである。幅広い粒度のデータを表現するためにはこの複合オブジェクトの考えが必要不可欠である。これについては次節で述べる。

3.2.2 複合オブジェクト

前述したようにソフトウェアデータベースでは従来の関係データベースのようにデータ構造が比較的簡単なものを扱うのではなく、主に構造を持つデータを扱うため、これらの複雑なデータを表現するための機能が必要である。その中でも重要な表現方法として複合オブジェクトがある。これは、複数のオブジェクトをまとめて１つのオブジェクトとみなし、ソフトウェアデータベースで頻繁に利用される階層構造を自然に表現することができる表現方法の１つである。

しかし、複合オブジェクトをデータベースの側から考えた場合、親子オブジェクトの操作に関していくつかの問題がある。その１つとして、子オブジェクトが欠如している場合の親オブジェクトの状態がある。あるべき子オブジェクトが１つでもないと親オブジェクトは不完全と見なされるのか、または全く子オブジェクトを持たない親オブジェクトは存在可能か、など事前に複合オブジェクトの意味的解釈を決定しておく必要がある。さらに重要な問題として、親オブジェクトが削除された場合の子オブジェクトの振舞いがある。子オブジェクトは親と共に削除されるべきなのか、または、親無しのオブジェクトとして維持されるのかは状況によってまちまちである。これらの問題はシステムの低レベルのデータ管理機構で支援する必要がある。

3.2.3 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。さらに、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このような修正を行なった理由がわかれば再修正にかかる手間も省けるであろうし、どうしても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[3] が必須である。

バージョン管理を実現する一番簡単な方法は全てのバージョンをそのまま保存する方法であるが、当然ながらこの方法は莫大なスペースを必要とする点で非効率である。そのためSCCS[4] やRCS[5] のように各バージョン間で差分をとる方法がある。テキスト形式のファイルの場合、隣接バージョンとの違いは局所的であり、そのため差分ファイルは小さい。しかしRCS やSCCS ではテキスト形式のファイルしか扱うことができない。ソフトウェアデータベースで扱うデータはテキストファイルの他にバイナリ形式のファイルも扱うことが多いため、それ専用のバージョン管理システムを用意する必要がある。ただし、バージョン管理に必要な基本的な機能は、差分をとるアルゴリズムが違うだけでテキスト形式でもバイナリ形式でも差異はない。

3.2.4 コンフィギュレーション管理

最終的な生成物を作成する場合、分割された各モジュールから必要なモジュールを集めて作成される場合が多い。各モジュールはさらに細かなモジュールに分割されているかも知れない。このようにそれぞれの部品から１つの製品を作るためには、その間の

関係、つまりコンフィギュレーション関係を管理する必要がある。この関係を管理することによって、例えば、あるモジュールを作成するために必要なものを開発者が陽に与える必要がなくなり、さらに、各構成要素（モジュールやファイルなど）のタイムスタンプ（最新の修正時間）を管理することで、1ファイルを修正しただけで全構成要素を処理する必要がなくなり、修正によって変更されるものだけが再構成されればよくなる。このようなコンフィギュレーション管理を行う既存のシステムとしては、UNIXでは著名な"make"がある。

3.2.5 参照、包括関係管理

ソースコードや文書内には、定義とその定義を使用している間の参照関係が多く含まれる。しかし、この関係を管理するためには細かいデータ粒度が要求される。例えば、C言語のソースコードの場合、関数定義部やグローバル変数を参照している部分を扱うためにはステートメント中の式までデータとして格納しなければならない。しかし、データ粒度が細かいほど細かな照会にも答えることが可能となる。包括関係の実現には前述した複合オブジェクトを扱う機能を利用できる。

3.2.6 排他制御

関係データベースにみられるような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、ソフトウェアデータベースでは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ので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。

また、複合オブジェクトに関する排他制御で複合オブジェクトをロックしてしまう場合、それを構成するオブジェクト全体をロックするのかどうかの問題がある。例えば、プロジェクト全体をロックしてしまうとあるファイルのあるトークンですら修正することもできなくなってしまう。

4 E-Rモデルに基づくソフトウェアデータベース

ソフトウェアデータベースを実現するに際し、さまざまなアプローチがあるが、本論文では実体と実体間の関連を記述するのに最適なE-Rモデルに基づいてソフトウェアデータベースの実現を考える。

4.1 E-Rモデルの採用理由

データベースは概略的に大きく2つに分けることができる。1つは、実際にデータの格納、アクセス、操作、管理などを行う制御部分と、もう1つは、格納されているデータのviewを与える表現部分である。もちろんE-Rモデルは制御部分と表現部分の2つを引き受けられるほど万能ではない。E-Rモデルは抽象度が高く、もともと概念設計のデータモデルとして適しているため、しばしばデータベースの表現部分に採用され、制御部分は既存のデータベースシステムを利用することが多い。データベースの表現部分には数あるデータモデルの中から選べたのだが、その中からE-Rモデルを採用した主な理由として次のようなものがある。

1. ソフトウェアデータベースでは実際に生成されるデータ（実体）とデータ間の関連の柔軟な記述が重要
2. 設計者にとっては概念スキーマを直接表現できる
3. 利用者にとっては概念スキーマを理解しやすい
4. ユーザインターフェイスとしてE-Rダイヤグラムを利用できる

4.2 E-Rモデルによる関係管理

関係データベースではその中に存在する複数のテーブルを自在にジョイントすることによって、さまざまな関係を定義することができ、しかもそれにより、ユーザ側にはなんの意味もないテーブルを作成することもできる。それだけ関係に対して自由度が高いということがいえるし、関係データベースの特徴の1つになっている。しかし、開発用言語に対する照会などにみられるように、ソフトウェアデータベースで出現する関係はある程度限定されたものである。仮にある言語に対して関係データベースのような自由度の高い照会に対応するためにはさまざまな関係をデータベース設計者が定義しておく必要が

あるし、もし関係定義がなかった場合にはユーザが定義できる方法を用意しなければならない。

ソフトウェアデータベースでよく出現する関係は3.1でも挙げたが、定義と使用の関係、包括（親子）関係、オブジェクトコードとソースコードのコンパイル関係、バージョン関係、コンフィグレーション関係などがある。これらはE-Rモデルでの関連にあたり、データベース設計者が概念スキーマをE-Rダイヤグラムで表現する場合の基本的な関連として定義され、図4のように関連ダイヤモンドとして表現することができる。

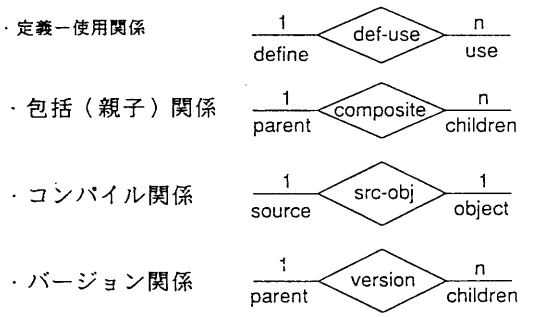

図4: ソフトウェアデータベースでの基本的な関連

例えば、データベース設計者がC言語でファイルと関数定義部を実体として定義したとする（図5）。

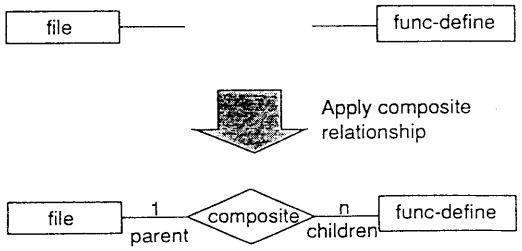

図5: ファイルと関数定義部の関連

この場合、関数定義部はファイルの構成要素となっているので包括関係の関連ダイヤモンドを使って2つの実体を結ぶ。さらに図5のファイルにバージョン関係を付け加えたい場合は図6のようになる。

このように、E-Rモデルでの関係管理を利用して

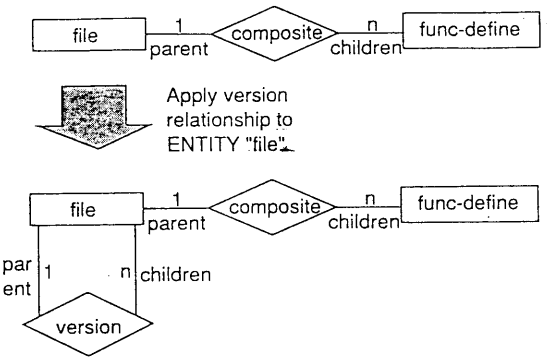

図6: ファイルとバージョンの関連

バージョン管理、コンフィギュレーション管理、参照、包括関係管理を記述することができる。

5 ソースコード照会データベース SQUDER

SQUDER(Source code QUery Database based on E-R model) は、ある言語のソースコードに対して照会を行うためのデータベースである。照会の対象となるデータ（関数定義部や変数宣言部など）をE-R風に記述し、それを基に作成されたデータベースとのインターフェイスライブラリを対象言語の処理系に組み込むことで実現している。

5.1 SQUDERの概要

SQUDERは以下のように3つの層から構成される。

ユーザ層	emacs-lispを使ったNEmacs、gcc
E-R Interface層	E-Rライブラリ
データベース層	INGRES

ユーザ層はデータベース利用者が操作する層である。E-Rインターフェイス層によって提供されるデータベースの操作コマンドを直接ユーザが、または応用プログラムが利用してデータの操作を行う。今回データベース設計者から与えられた操作コマンドはCソースコード内の参照関係に関するものであり、その関係データの入力はGNUのcc(以下gcc)が、データの出力は日本語EmacsであるNEmacs [6] によって行われる。E-Rモデルで多用されるE-Rダイヤグラムはユーザの理解を助けるのに絶大な効力を発揮する

ことを利用して、将来この層にgraphicalなインターフェイスを付け加えることによって、E-Rモデルでのよりよいインターフェイスが実現するであろう。

E-Rインターフェイス層はユーザ層とデータベース層の橋渡しをする層で、ユーザが直接データベース層を操作しないように設計される。つまり、ユーザ層から見たE-Rインターフェイス層はE-Rモデルに基づいたデータスキーマに関する操作を提供しており、データベース層からみたE-Rインターフェイス層はデータベース層で採用されているデータスキーマに関して操作を行う層である。この層は詳しくは後述する。

データベース層はE-Rインターフェイス層からのデータを実際に格納する層である。今回この層には4.3BSDの配布テープにあるINGRES[7]を採用した。このINGRESはPDSの関係データベースであり、データベース操作言語(DML)にEQUELを使っている。

5.2 C言語における照会対象のE-Rによる記述

C言語の全ての関係を取り上げて説明すると複雑になるので、ここではCでの照会対象として最も直感的に理解しやすいものとしてファイル、関数定義部、グローバル変数定義部とその包括関係を取り上げる。これらをE-Rダイヤグラムで表現したものを図7に示す。一見してわかるように設計者の構想をそのまま下書きしたようなものである。これに属性の型やPrimary Key(PK)を付け加えて具体的に記述したものが図8である。これがSQUDERの基本となるデータの記述部であり、この記述をもとにE-Rインターフェイス層が作成される。

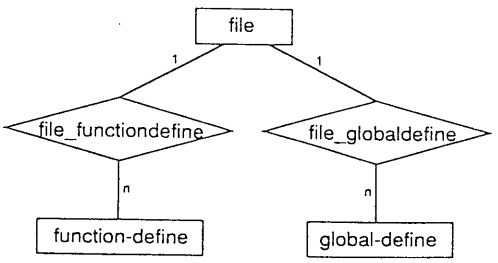

図7: SQUDERのE-Rダイヤグラム

```
DECLARE         inode           INT     5
                filename        CHAR    64
                funcname        CHAR    32
                owner           CHAR    8
                location        INT     4
                modifytime      INT     8
                type            CHAR    32
                strageclass     CHAR    8
                gloname         CHAR    32

ENTITY          file
        ATTR    inode,filename,owner,modifytime
        PK      inode

ENTITY          funcdef
        ATTR    funcname,inode,type,strageclass,location
        PK      funcname,inode

RELATION        file_funcdef
        ATTR    inode,funcname
        PK      inode/1,funcname/N

ENTITY          glodef
        ATTR    gloname,inode,type,strageclass,location
        PK      gloname,inode

RELATION        file_glodef
        ATTR    inode,gloname
        PK      inode/1,gloname/N
```

図8: SQUDERの記述部

5.3 SQUDERのデータ入力部

以上で示したデータの入力はCのソースコードをgccでコンパイルすると同時に行われる。実際は、gccのオブジェクトとE-Rライブラリをリンクして作成されたgccを使用する。このE-Rライブラリは前ページのE-Rの記述部から作成される。これらの様子を図9の概略図で説明する。

この図で、sample.erから'ertoq'というツールを用いてsample.qが作成されている。この'ertoq'というツールは、E-R記述をもとに、各ENTITYとRELATIONにおけるINGRESの基本操作をあつめたライブラリソースコードを生成するものである。基本操作には4つあり、ENTITY(RELATION)の獲得、追加、削除、確認である。例えば、sample.erの'ENTITY file'から、ERaddENTITYfile(), ERgetENTITYfile(), ERdeleteENTITYfile(), ERcheckENTITYfile()という4つのCの関数が生成される。図8のE-R記述部から生成されるものの一部分を図10に示す。'##'で始まる行はINGRESのDMLであるEQUELのソースコードである。これをEQUEL(データベース操作言語と同名のツールである)というツールがCのソースコードに変換する。これをCコンパイラでコンパイルしたものがE-Rライブラリとなる。

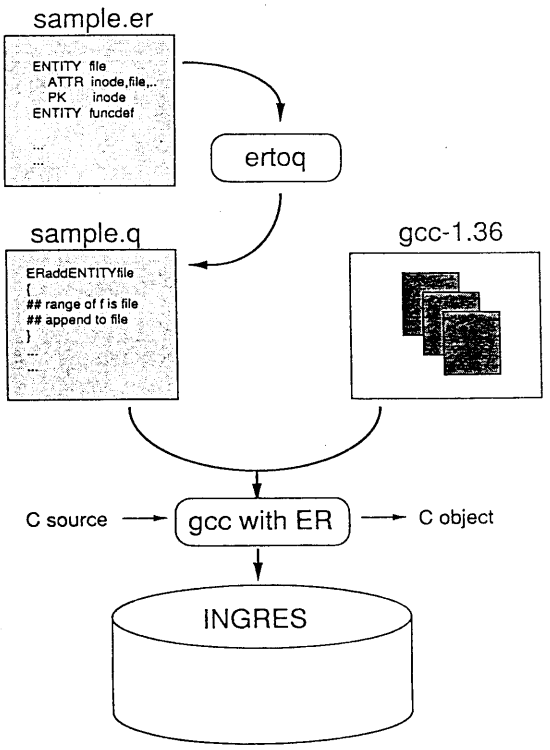

図 9: SQUDER のデータ入力部

```
## int  inode;
## char filename[64];
## char funcname[32];
## char owner[8];
## int  modifytime;

ERaddENTITYfile ( inode, filename, owner, modifytime )
int inode ;
char *filename ;
char *owner ;
int modifytime ;
{
## range of f is file
## append to file (
##       #inode = inode,
##       #filename = filename,
##       #owner = owner,
##       #modifytime = modifytime )
}

ERgetENTITYfile ( inode )
{
## range of f is file
## retrieve (
##       inode = f.#inode,
##       filename = f.#filename,
##       owner = f.#owner,
##       modifytime = f.#modifytime
## ) where f.#inode = inode
## [
:
## ]
}
```

図 10: ertoq の出力例

5.4 SQUDER のデータ出力部

NEmacs の emacs-lisp を使用し、実際にデータを SQUDER から抽出して定義部を表示している様子を図 11に示す。定義部を見たいと思う関数名やグローバル変数名の上にカーソルを移動し、コマンド sd-do-search を実行する。その定義部が 1 つであれば、そのファイルのウィンドウを開く。複数ある場合、選択肢を表示し、表示したいファイルを選択させるようになっている。ちなみにこの emacs-lisp は 198 行、14 関数で構成されている。

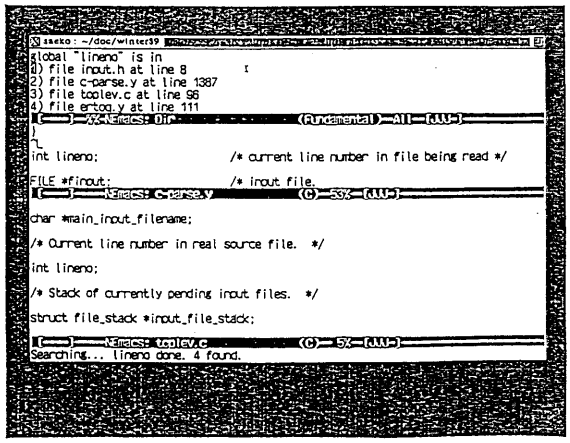

図 11: SQUDER の NEmacs によるデータ出力例

5.5 SQUDER の性能評価、問題点

SQUDER のデータの入力は現在のところ gcc で行なわれている。以下に、オリジナルの gcc と E-R ライブラリとリンクされた SQUDER 用の gcc の比較を、C ソースコードのコンパイル時間で行なった結果を示す[2]。

	User time	System time	Total
普通の gcc	24.0s	1.7s	25.7s
SQUDER 用の gcc	107.8s	59.8s	167.6s

ちなみに比較に使用したソースコードは約 3900 行、46 関数のかなり大きなものである。結果から、User Time が普通の gcc に比べて 4 倍にも関わらず、SQUDER 用の gcc は System Time にかなりの時間がかかっていることが分かる。これは、SQUDER 用の gcc

[2]計測は SUN3-60 で行なったものである。

の内部でINGRESが起動されており、このINGRESがかなりのシステムコールを実行するためである。

6 E-Rモデルに基づくソフトウェアデータベースの問題点、将来

本論文ではソフトウェアデータベースを実現するに際し、E-Rモデルによるアプローチを採用した。本章では、このアプローチに関する問題点を指摘しE-Rモデルによるソフトウェアデータベースの今後についての考察を加える。

6.1 問題点

6.1.1 不自然な階層構造の表現

一般的に階層構造を図で表現する場合、図12(a),(b)のように木構造や包括関係で表現するのが普通である。しかし、E-Rダイヤグラムでは同じ性質を持つ実体は同じ1つの実体集合で表現されるため、図12(c)のように一見して階層構造を表現しているとは思えないような表現になってしまう。この場合、親を持たない実体が木構造でいうところの根になり、子を持たない実体が葉となる。

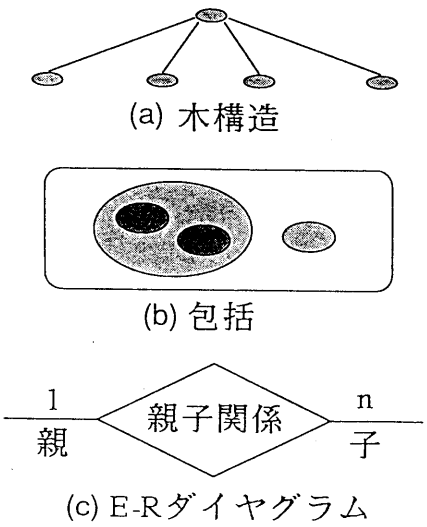

図12: 階層表現(a)木構造(b)包括(c)E-R

図12(c)での関連は子供の存在は親の存在に依存しているので、E-Rモデルでいう弱関連である。

6.1.2 データの不完全性の問題

複合オブジェクトの節(3.2.2)でも述べたが、子オブジェクトが欠如している場合の親オブジェクトの状態や、親オブジェクトが削除された場合の子オブジェクトの振舞いの問題に関してはまだ解決していない。突き詰めて考えるとこの問題はデータベースの制御部分、つまり、データを直接操作する低レベルの段階で解決しないと根本的な解決は望めないかも知れない。例えば、階層構造を持つデータがあるとする。このデータ中の木構造で節にあたる部分が削除されたとすると、そこについていた子をどう処理するべきであろうか。その処理方法としては以下のように2つ考えられる。

- その子以下の部分木も削除する
- そのままその子を根とする部分木として残す

前者はデータ間の関係が包括関係である場合や、複合オブジェクトの場合にあたる。例えば、Cソースファイルが削除されればその中身である関数定義部やグローバル変数定義部も削除されるべきである。後者はバージョン関係の場合にあたる。バージョン関係を表す木の節を削除する操作はかなり危険な操作ではあるが、謝った操作で消してしまった場合、それ以下の部分木まで削除する必要はない。このように階層構造データの削除に関する操作に限っても、3.1で述べたソフトウェアデータベースに存在するデータ間の関係毎にその操作内容が異なってくる。

6.2 将来

6.2.1 E-Rダイヤグラムによるインターフェイスの必要性

従来のデータベースのユーザインターフェイスはおもにテキストや表形式中心で行われてきた。関係データベースのように扱うデータの型が比較的単純なものであればテキスト形式で十分であったが、ソフトウェアデータベースに限らず扱うデータの構造が複雑な場合、ある程度図式化することでユーザの理解を助けることができることは言うまでもない。E-Rモデルのダイヤグラムは概念的、抽象的にデータを表現するための形式的手法を提供している。

このE-Rダイヤグラムをデータベースのインターフェイスに使用することで、特に、ハイパーテキスト的なデータベース内のナビゲーションが可能になる。このデータベースのナビゲーション、また、ブラ

ウジング機能はデータベース内にどのようなデータが存在するのか、データ間にどのような関係があるのかを理解するためにはなくてはならない機能である。近年、ビットマップディスプレイを有するワークステーションの普及に伴い、グラフィカルなインターフェイスを駆使したシステムをしばしば見かけるようになった。E-Rモデルに基づくデータベースでは、E-Rダイヤグラムという図的表現方法を利用して、今までのデータベースにはなかったユーザフレンドリなインターフェイスを提供することが望まれる。

6.2.2 ソフトウェアプロセスとの融合

ソフトウェアプロセスとは、L. Osterweilが提唱したもので[8]、ソフトウェア開発工程自身をソフトウェアと考えて、開発工程自身をプログラミングしようというものである。このソフトウェアプロセスとソフトウェアデータベースは独立に存在しているのではなく、ソフトウェアプロセス内で生成されるデータ全てがソフトウェアデータベースに格納され、ソフトウェアデータベース内のデータはソフトウェアのライフサイクルの上流である仕様作成から、下流の保守までの間で動的に変化していくのである。現在のところ、ソフトウェアプロセスとソフトウェアデータベースの融合に関してはW. RiddleがSDA (Software Designer's Association) で提案している[9]。このSDAはソフトウェア設計環境を、ソフトウェアの設計手順を示した設計プロセス記述、ソフトウェア開発のためのツール群、そしてソフトウェアデータベースという3つに分けて捉えるものである。このようにソフトウェアデータベースもソフトウェア設計環境の一部分となり、ソフトウェアプロセス、開発環境などのその他の部分と共存、統合していくことで、良質のソフトウェアの開発に威力を発揮することであろう。

7 おわりに

今回は、とりあえず実働するものを作成しようということで、ソフトウェアデータベースにおけるプログラムの照会機能のプロトタイプとしてSQUDERを実現した。照会のみなので関係データベースのINGRESで実現できたが、これをソースコードや文書を実際に格納できるソフトウェアデータベースに拡張するにはINGRESでは不可能である。そのため、オブジェクト指向DBMSなどを利用する方法が考えられるがやはり問題が残る。やはりソフトウェアデータベースの要求にあったDBMSの実現が待たれるところである。

最後になりましたが、東京工業大学情報工学専攻徳田研究室の皆様、日立システム開発研究所の大槻繁さんに感謝致します。

参考文献

[1] Peter Pin-Shan Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1), March 1976.

[2] Philip A. Bernstein. Database system support for software engineering. In *Proceedings of the 9th International Conference on Software Engineering*, March 1987.

[3] R.H Kats, E. Chang, and R. Bhateja. Version Modelling Concepts for Computer-Aided Design Databases. In *Proceedings on ACM SIGMOD International Conference on Management of Data*, pages 379–386, May 1986.

[4] Marc J. Rochkind. The Source Code Control System. In *Proceedings on First National Conference of Software Engineering*, pages 37–43, 1975.

[5] Walter F. Tichy. RCS – A system for version control. *Software – Practice and Experiences*, 15(7):637–654, July 1985.

[6] Richard Stallman. *GNU Emacs Manual*. Free Software Foundation, 675 Massachusetts Avenue Cambridge, MA 02139, sixth edition, version 18 edition, March 1987.

[7] Joe Kalash, Lisa Rodgin, Zelaine Fong, and Jeff Anton. *INGRES Reference Manual*. version 8 edition, May 1989.

[8] L. Osterweil. Software Processes are Software Too. In *Proceedings of the 9th International Conference on Software Engineering*, 1987.

[9] W.E. Riddle. Software Designer's Associates: a Preliminary Descriptions. In *Proceedings of 20th HICSS*, pages 371–381, 1987.