1R-3

統合プログラミング環境（２）開発支援データベース

小林　茂　　松本憲幸　　落合正雄
（株）東芝

1．まえがき

ソフトウェア開発において取扱う様々な対象を管理するためのデータベースとして、ＰＤＤ（Program Data Dictionary）を開発した。ソフトウェア開発用データベースに対する考え方と、ＰＤＤの概要について説明する。

2．ソフトウェア開発におけるデータベース利用の目的

一般に、ソフトウェア開発においては非常に多くのソフトウェアオブジェクト（以下、単にオブジェクトと呼ぶ）が操作対象となる。たとえば、データファイルとその中のレコード、プログラムファイル、プログラムモジュール、あるいはそれらを処理するコマンドなどである。

現状の開発環境では、これらのオブジェクト、およびその間の関係が開発者に見えている。そして、開発者はそれらについて憶え、コマンドを実行したときにそれらがどのように影響されるかを意識しながら作業をしなければならない。また、オブジェクト情報を参照する場合にも、その提示形式はコマンドによってまちまちである。

オブジェクト情報のデータベース化により、開発者のオペレーションに一貫性を持たせ、かつ、真に必要とする情報のみを提示することが可能となる。また、コマンド間での情報の相互利用による自動化や、開発者に必要のない情報の隠蔽なども容易になり、開発者の負荷を軽減し、開発を効率化することができる。

3．ＰＤＤの特徴

ＰＤＤは、ソフトウェア開発支援という特定の用途を想定しているために、一般のデータベースとは機能上のウェイトの置きかたが異なっている。設計上、特に重視したのは以下の点である。

(1) 少ない容量（ファイルサイズ、メモリサイズとも）で、かつ高速に実行できること。
(2) オブジェクトについて、的確な表現ができること。
(3) 開発プロジェクトによるデータベース利用をサポートすること。
(4) プログラムからのデータベースアクセスの自由度が高いこと。

容量と速度に関しては、最もレコード件数の多い例と思われるクロスリファレンス情報の場合（５０Ｋ件程度）で、最終的に、ファイルサイズ５００ＫＢ、登録時間１０秒以下、標準的な検索の時間が１秒以下を達成する見通しである。
項目(2)～(4)については、次節で述べる。
上記の点とは逆に、以下の点に関して優先度を下げている。

(5) データ保全機能は性能上の負担とならないレベルのものとすること。
(6) レコード形式の変更はプログラムの修正を必要とすること。

以上の選択は、開発用のデータベースは、データの損傷や構造の変更が生じてもオブジェクトの実体から比較的容易に再構築できること、および、コマンド類の扱うレコード形式が変更されることは少ないと思われることによる。

4．ＰＤＤの機能

4．1．機能構成

ＰＤＤを利用した開発環境の機能構成を図１に示す。
ＰＤＤデータベースは、ファイルシステム上の一つのディレクトリとして実現される。アクセスメソッドは、このデータベースにプログラムからアクセスするためのインタフェースルーチン群である。開発環境に含まれる各種のコマンドは、アクセスメッソドを用いて相互に情報の登録・参照を行う。ＰＤＤ－Ｓｈｅｌｌは、これらのコマンド群を統合するユーザインタフェースであり、コマンド実行に伴う補足的な情報修正を行うほか、データベースの会話的アクセスや、保守のための機能も含んでいる。

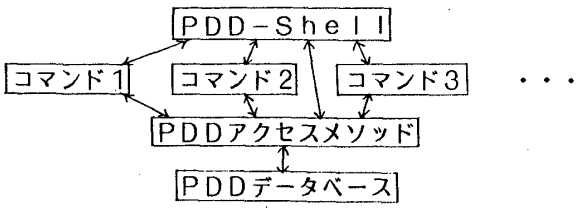

図１．ＰＤＤの機能構成

A Design of Database for Integrated Programming Environment
Shigeru Konayashi, Noriyoshi Matsumoto, Masao Ochiai
TOSHIBA Corp.

4．2．表現能力

データベースが開発者への情報提示や開発作業の自動化などにおいて十分に機能するためには、現実世界のオブジェクトやそれらの間の関係について、的確に表現できる必要がある。ＰＤＤでは、これらをオブジェクト型のレコードと、関係型のレコードによって表現する。レコードの論理的な構造は次のようになる。

オブジェクト型レコード：　オブジェクト型名（属性値１，属性値２，・・・）
関係型レコード：　関係型名（オブジェクト１,オブジェクト２，・・・）

下の例からもわかるように、オブジェクト型のレコードを用いることによって、現実世界の状態をより正確に記述できる。

関係型レコードのみによる表現の例

```
refer( "file_a.c", "i", 20, "def" )      /* file_a.c の 20行目で i を定義 */
refer( "file_a.c", "i", 90, "def" )      /* file_a.c の 90行目で i を定義 */
refer( "file_a.c", "i", 100, "ref" )     /* file_a.c の 100行目で i を参照 */
```

オブジェクト型レコードを用いた表現の例

```
OBJ_ID1 = id( "i", "var" )
refer( "file_a.c", OBJ_ID1, 20, "def" )
OBJ_ID2 = id( "i", "var" )
refer( "file_a.c", OBJ_ID2, 90, "def" )
refer( "file_a.c", OBJ_ID2, 100, "ref" )  /* 90行目で定義された i を参照 */
```

4．3．データ管理能力

開発プロジェクト全体で一つのデータベースを共有するような場合、データベース内の全ての情報が常に操作対象になるわけではない。情報を適当にグループ化し、書換えや参照の対象領域を限定することができれば、誤って他の開発者のデータを破壊することを防いだり、検索を効率化したりすることが可能である。このようなデータ管理を実現するために、ＰＤＤは「サブデータベース」という機構を組込んでいる。

サブデータベース内でも、元のデータベースで定義されている全てのレコード型を使用できる。一つのデータベースが複数のサブデータベースを含むことも可能であり、また、サブデータベースの中に、より下層のサブデータベースを作ることもできる。サブデータベースにはそれぞれ所有者、および各開発者に対するアクセス権（検索権、変更権）を設定される。（図２）

検索の対象とするサブデータベースの集合、および変更の対象となるサブデータベースは、アクセス時に自由に選択できる。また、必要なサブデータベースのみを指定して、データベースをコピーする機能がある。こうしたサブデータベースの機構は、共有のインタフェース情報の保護に利用できるという点でも、開発プロジェクトの支援に適している。

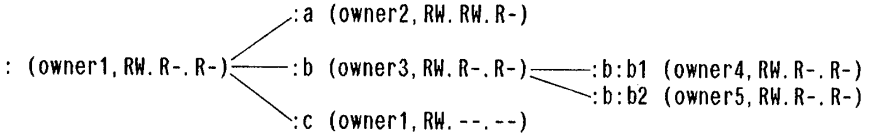

図２．階層化されたデータベースの例

4．4．アクセスメソッド

ＰＤＤは、アクセスメソッドを用いて、プログラムから手軽に利用できる。主要なメソッドを以下に挙げる。

(1) データベースの生成　　credb()
(2) アクセスの開始・終了　　open()→<アクセス>→close()
　（大部分のデータベースアクセスは、open と close の間で行う。）
(3) レコード型の定義　　cretbl()
(4) レコードの登録　　crercd()
(5) データベースの検索　　select()→<条件>→fetch()

検索の条件として与えられるのは、主に特定のフィールド値を持ったレコードパターンの列であり、これらはＡＮＤ条件として働く。そのほかに、ＯＲやＮＯＴ、およびフィールド値の比較を記述できる。また、「あるモジュールから直接・間接に呼ばれている全モジュール」といった検索のために、再帰的な条件の記述手段も用意されている。

5．あとがき

ソフトウェア開発支援用データベース、ＰＤＤについて説明した。ＰＤＤはオブジェクト管理のために強化された多くの機能を含んでいるが、なお改善の余地がある。例えば、オブジェクトを人間の直観により近い形で表現するために、レコードのフィールド値として構造体やリストを追加する、等を考えている。（現在は数値・文字列値のみ。）

今後、ＰＤＤを利用した開発環境を充実しながら、その高機能化・高性能化を進めていく予定である。