データベース・システム 85－7
プログラミング－言語・基礎・実践－ 4－7
（1991. 10. 4）

パネル討論：

永続的プログラミング言語とオブジェクト指向データベース

司会　　：安村通晃（慶応大学）
パネリスト：大堀淳（沖電気）、小野寺民也（日本ＩＢＭ）、鶴岡邦敏（日本電気）、
布川博士（東北大学）、牧之内顕文（九州大学）　［五十音順］

近年、次世代データベースを目指したオブジェクト指向データベースの研究開発が進展しており、そこではデータベース言語とプログラミング言語との融合が議論されている。また一方で、プログラミング言語に永続性を導入しようとするいわゆる永続的プログラミング言語の研究も盛んになりつつある。双方の動きが相まって、今後のプログラミング言語とデータベースとの位置づけに関して、新たな枠組みが形成されるかに見える。本パネル討論では、こうした状況を踏まえて、プログラミング言語側・データベース側の双方から見た他方への要請や問題点、及び両者のインタフェースに関わる課題等を議論し、今後の方向を探る。

Panel Discussion:

Persistent Programming Languages and Object-Oriented Databases

Chaired by: Michiaki YASUMURA (Keio University)
Panelists : Atsushi OHORI (Oki Electric)
Tamiya ONODERA (IBM Japan)
Kunitoshi TSURUOKA (NEC Corporation)
Hiroshi NUNOKAWA (Tohoku University)
Akifumi MAKINOUCHI (Kyushu University)

Recently, research and development of object-oriented database systems are going on for the next generation databases, where the unified architecture of database language and programming language is discussed. On the other hand, as the introduction of persistence to programming languages, research interests are growing in persistent programming languages. It seems that these trend together will form a new paradigm for the positions of programming language and database. This panel discussion, based on the observations above, states requirements for programming languages and databases viewing from the other side, discusses their inteface issues, and finds the future directions.

データベース・システム 85－7－1
プログラミング－言語・基礎・実践－ 4－7－1
（1991. 10. 4）

パネル討論：永続的プログラム言語とオブジェクト指向データベース

型理論に基づく次世代データモデル実現の可能性

大堀 淳　　沖電気工業（株）関西総合研究所

1 次世代データモデルのためのパラダイム

近年オブジェクト指向データベースが次世代のデータベースシステムとして注目を集めているが、それについての形式的定義は、筆者の知る限り未だ存在しない。最近オブジェクト指向データベースの「満たすべき条件」に関する幾つかの議論があるが、何を持ってオブジェクト指向データベースと呼ぶかは未だに議論の対象であり、何がこれまでと違って本質的に革新的なのかについても十分な説明が与えられているとは言い難いと思われる。新しいデータモデル（オブジェクト指向であれ何であれ）が、関係データベースシステムが経験したような実用化技術の蓄積を享受し、次世代のデータベースシステムの基礎となり得るためには、そのモデルに対して、関係データモデルの記述言語が持っている形式的で整合性のある意味論に対応するような形式的基礎が確立されることが必須であると考える。これが本パネルでの筆者の基本的な立ち場である。

データベースの新しいモデルの可能性は一つとは限らず、次世代データベースのモデルの形式的基礎を与えるためのアプローチもまた複数存在し得る。そのなかで特に重要でかつ有望と考えられるものは、プログラム言語とデータベース両方の概念を統合した理論の構築である。オブジェクト指向データベースは、少なくとも歴史的には [2]、言語システムとして提案されたものであり、オブジェクト指向データモデルがあるとすれば、それはプログラム言語とデータベースの概念を統合したものとなると考えるのが自然である。また、そのような統合されたシステムは、単なるデータ検索システムにとどまらず、次世代の情報処理システムの中核とり得ると期待できる。

この統合の基礎となり得る理論の一つとして、筆者は、プログラム言語の型理論（例えば [4] を参照）が有力であると考える。型理論は、汎用の計算能力と複雑なデータ構造を統一して制御する理論であり、プログラム言語、データベース、オブジェクト指向等の諸機能の統合を目指す次世代データベースの基礎理論として適していると考えらるからである。ここで留意していただきたい点は、本稿では型を、プログラム言語理論の歴史の中で洗練され確立した概念（およびその一般化）として用いるという点である。オブジェクト指向データベースの文献等では、時おり、「型」や「クラス」およびそれらの区別に関する議論を見かけるが、それらの議論が有意義に行なえるためにも、この厳密な態度は重要であると考える。さらに、それら直観的な「型」や「クラス」、「スキーマ」等の概念間の関係を明らかにし、それらを統合する枠組としても、形式的定義に基づく型理論は有用であると信じる。

型理論にはそれ特有の弱点があり、オブジェクト指向データベースや種々の次世代データベースで提唱されているすべての機能を自然に表現する見込みは未だ立っているわけではない。しかし筆者は、近年の型理論の急速な発展を考慮すれ時、型の概念をさらに洗練し拡張することによって、多くの未解決に問題も解決される可能性が十分にあり、型理論に基づくデータベースの実現は一つの有望な研究テーマであると信じる。

本パネルでは、以上の立場から見た永続言語の特徴、永続言語へデータベース機能、およびオブジェクト指向プログラミングの機能を導入する上での課題についての筆者の考えを論じる。

2 型理論からみた永続言語の本質

永続データという用語は、個々のプログラムの実行とは独立に、長期にわたって存在するデータを意味する。しかし型理論から見たデータの永続性の本質は、その生存期間の長さではなく、それらがプログラムの静的環境の外部にあるデータという点にある。（この点で、永続データは例えば、分散環境下の他のシステムのデータなどと似た性質を持つ。）この外部性のため、プログラム言語の静的型システムは、永続データに関する型情報を静的に決定することができず、動的な型のチェックがどうしても必要である。永続データを扱える型システム設計の要点は、静的型システムの種々の性質を損なうことなく、動的型チェックの機構を導入することである。PS－algol[3] をはじめとする永続プログラム言語の研究の最大の貢献は、この機構を組織的に型つきプログラム言語に導入したことでる。それは、永続データを、型情報を含んだ特殊な型を持つデータとして型システムに導入し、永続データを扱う際に必要な動的型チェックを局所化する機構である。このようなデータは、型理論の枠内では dynamic 型を持つデータと呼ばれ、文献 [1] で形式化されている。

以上の諸研究によって明らかになった重要な点の一

つは、型つき言語にとって、データの永続性は他の性質との独立性が高いという洞察である。永続プログラム言語の研究者によって提唱されたスローガン「直交永続性」は、この性質を根拠とするものである。このことは、データベースやオブジェクト指向プログラムの種々の機能が型つき言語において形式化できれば、それらと永続データを取り扱うメカニズムとの統合は比較的容易であることを意味する。

3 永続言語の次世代データモデルへの拡張

以上論じた永続データを取り扱う機能は、プログラム言語のデータ操作能力を外部の永続データへ拡張する機能であり、それ自体ではデータベースシステムに必要な機能を実現することはない。型理論に基づくデータモデル実現のためには、型システムを拡張して、データベースシステムにとって必要な機能を表現可能にする必要がある。このためのに解決しなければならない課題は数多く存在するが、その中で特に重要と考えられる点には以下のものがある。

1. 複合オブジェクト操作システム
   次世代データベースにとっても、関係代数に匹敵するエレガントで強力な操作系が望ましいのは明らかである。オブジェクト同一性の機能を含まない、値としての複合オブジェクトの操作システムを形式的に定義する試みはいくつかあるが、オブジェクト同一性の機能を持つ複合オブジェクト操作系に対する、一般性のあるエレガントな定式化は、未だ成功していないようである。これは、最近話題となっているオブジェクトの「ビュー」の機能を実現する上でも重要な研究課題である。

2. データベースの設計理論
   関係データベースに比べて飛躍的に複雑さが増すと予想される次世代データベースにとっては、より緻密な設計理論が必要になってくることは明らかである。この場合もやはり、オブジェクト同一性の機能を持つ複合オブジェクトに対する実用性のあるしかも数学的にエレガントな設計理論の確立が課題である。

   関係データベースにおいて、エレガントな操作系や設計理論が可能であったのは、それが数学的基礎の基に成り立っていることによることはあきらかであり、以上二つの目的を達成するためには、データモデルの形式的基礎が必須である。

3. 大容量型 (bulk data types) についての理論
   データベースは、型理論では、集合やリスト等の大容量型を持つデータと見なせる。これら大容量型の性質に関する研究は、その重要性にも関わらず、これまであまりなされていない。

4. オブジェクト指向プログラムの機能の導入
   オブジェクト指向プログラミングの諸概念を型システムの中で表現する研究は、現在、型理論の研究でもっとも盛んに研究されているテーマの一つである。これらの研究成果は、次世代データベースシステムにとっても極めて有用である。

5. 実装理論
   データモデルの理論は効率的でかつ信頼性のある情報処理システムを実装するための理論であり、(何らかの意味で) 実装技術に貢献し得ない理論は意味に乏しい。型理論はこの分野でも大きな貢献をする可能性を持っていると信じる。これは、現実からかけはなれた根拠に乏しいこの主張と、受けとられかねないので、筆者の最近の研究であるが、敢えて例をあげることにする。文献 [5] では、レコードのラベルのアドレス情報 (あるいはオブジェクトに対するメソッドのアドレシング) を静的に決定する方法を与えている。この方法は、最新の型推論理論によって可能になったものである。

4 結論

型理論によりるオブジェクト指向プログラミングの諸概念やデータモデルの概念の基礎づけの研究は始まったばかりであり、実現されていない課題は多く存在する。しかし筆者は、型理論が、プログラム言語やオブジェクト指向の考えを統合した次世代のデータモデルを設計するうえでの重要な理論になり得ると信じている。これまで型理論は、データベースの研究者からは注目されることはほとんどなかったが、このパネルを通じて一人でも多くのデータベースの研究者がプログラム言語およびその型理論に関心を持たれることを、またプログラム言語の研究者がデータベースに関心を持たれることを期待したい。

参考文献

[1] M. Abadi *et. al.* Dynamic typing in a statically-typed language. In *Proc. ACM POPL*, 1989. [2] G. Copeland and D. Maier. Making smalltalk a database system. In *Proceedings of ACM SIGMOD*, pp. 316–325. ACM, June 1984. [3] M.P. Atkinson *et. al.* An approach to persistent programming. *Computer Journal*, 26(4), 1983. [4] J.C. Mitchell. Type systems for programming languages. In *Handbook of Theoretical Computer Science*, chapter 8, pp. 365–458. MIT Press/Elsevier, 1990. [5] A Ohori. A compilation method for implicitly typed polymorphic record calculi. In *Proc. ACM POPL*, 1992 (to appear).

データベース・システム 85－7－2
プログラミング－言語・基礎・実践－ 4－7－2
（1991. 10. 4）

Main Memory OODB

小野寺 民也
日本アイ・ビー・エム 東京基礎研究所

1 緒言

オペレーティングシステムやコンパイラといったシステムプログラムの分野でも、「データベース」は古くから重要な役割を果たしている。が、多くの場合、これらの「データベース」は、アドホックな方法で実現されてきた。長い歴史を持つ正当派データベースの研究の充実した成果を採り入れることにより、系統的な方法で実現できるはずである。

このようなシステムプログラムに現れるデータベースに対しては、主記憶型データベース（Main Memory Database、MMDB）[1] として実現される OODB が有効であると思われる。これが、本稿の主題である。OODB の定義として、ここでは、単純に $OODB = Persistent\ Objects + Atomic\ Transaction$ という立場をとることにする。次の2節および3節で、この2つの概念を「アドホックに実現された例」とともに簡単に説明する。そして、4節で「MMDB による OODB の実現」について考える。

2 Atomic Transaction

OODB に限らず、一般に DBMS の提供する重要な機能のひとつは、原子トランザクション（atomic transaction、以下、単にトランザクション）であろう。トランザクションは、データベースに対する処理の単位であり、これが原子的であるとは、可直列性 (serializability) と可復旧性 (recoverability) を有するということである。可復旧性について詳述すると、たとえトランザクションが（なんらかの理由で）完走できなかったとしても、データベースを中途半端な状態にしておかないということである。即ち、データベースはそのトランザクションが開始されたときの状態に復旧されるのである。

ここで、Version 7 あたりの UNIX での「データベース」のインプリメンテーションを考えてみる。通常、UNIX では、/etc/passwd のように「データベース」はファイルとして実現されている。「データベース」を更新するようなトランザクションは、エディタあるいはフィルタを用いて、ファイル全体を書き換える。このとき、原子性は次のようにして達成されている。

- トランザクションの可直列性は、排他ロックを用いることにより保証される。代表的な排他ロックは、システムコール creat によってロック用のファイルを作成することである。
- ソフト的な障害からの可復旧性は、1) 最初に、「データベース」のコピーをとり、2) 更新はコピーに対して行ない、3) 最後に、mv コマンドなどでファイルを rename することにより、瞬時に「データベース」を書き換える。—という手順で達成される。
- ハード的な障害からは、バックアップ・コピーからデータベースを復旧する。

3 Persistence

あるデータが persistent であるとは、データがプロセスの境界を越えて存在し続けるということである。そして、persistent objects というときの objects とは「プロセスの仮想記憶（とくにヒープ）に存在する、構造あるいは型を持ったデータ」を指す—と考えてよいであろう。

persistent objects の涙ぐましい実現例を言語処理系にみることができる。コンパイラ・プロセスがソースコードを処理しているとき、処理の進行とともに、仮想記憶上に数多くのシンボルが生成されていく。一方、デバッガ・プロセスは、これらのシンボルのもつ型情報やスタックフレーム上でのオフセットなどの情報を必要とする。コンパイラ・プロセスのシンボルはデバッガ・プロセスまで生き延びなければならない。通常、これらのシンボルは、コンパイラによってオブジェクトコードに encode された形で埋め込まれ、persistent なものになる。

4 Main Memory Database

システムプログラムで現れるデータベースの特徴のひとつは、その大きさである。たとえば、我々の現在使っているマシンの/etc/passwd の大きさは1K バイトにも満たないし、我々のLAN 全体のネームサーバーになっているマシンの/etc/hosts の大きさは 15K 足らずである。また、我々の使っているソースコード・ブラウザが管理するデータ量は、たとえば約3 万行のInterViews クラスライブラリ (バージョン 2.6) を最も詳細なレベルでブラウズするときで約4.5M である。

このような「小さな」データベースは、MMDB として簡単にかつ効率よく実現できるのではないかということが期待されている。実際、「ネームサーバのようなデータベース」を MMDB により実現する手法が、すでに提案されている [2]。ここで、「ネームサーバーのようなデータベース」とは、1) 大きさが高々10M で、2) トランザクションがシングルショットで、3) 更新トランザクションの発生が1 日10K 回程度でバースト時でも高々10 回/秒ほどであるものと定義されている。

MMDB の眼目は、データベース・サーバのプロセスの主記憶上にデータベース全体を (少なくとも論理的に) 展開することにある。最大の利点は、データへのアクセスが極めて速くなることで、そのコストは、アクセスパターンに依存するが、ページフォルトの回数に等しい。

原子トランザクションの実現のされ方につて考えてみると、先の [2] では、データベースの世界ではよく知られている、チェックポイントとログを用いる方法が使われている。更新するトランザクションは、ログを安定記憶域 (stable storage) にその都度書き出すことになる。このオーバーヘッドが小さくないので、対象となるデータベースは条件3) を満たす必要があるわけである。この条件を満たさないデータベース、即ち、シンボル表を管理するデータベースのように更新トランザクションが頻繁なデータベースについては、現在我々のグループでも研究中で、更新トランザクションに対して新しいサーバ・プロセスをフォークする方式を検討しているところである。

ここで重要なのは、主記憶上にデータベースを展開することによって、OODB における「2 次記憶管理」や「クラスタリング」といった問題が、仮想記憶における「局所性 (locality)」の問題に還元できることである。とすると、データベースと同様長い歴史をもつ、仮想記憶の研究の成果が利用できることになる。ワーキングセット理論、コピー方式のガーベージコレクタによる仮想記憶域の圧縮といった古くて新しい研究 (たとえば、[3]) が大いに貢献すると思われる。

5 結言

システムプログラムのデータベースは、バンキング・システムや JR の発券システムのように巨大なものではない。また、「OODBS 宣言」が提示するすべての条件を満たすような強力なものである必要もない。「小さな」データベースで十分なのである。また、今日では、主記憶が数百メガのワークステーションはあたりまえのものとなっている。この2 つの事実から考えると、MM-OODB というアプローチは有望であると思われる。それは、ともに長い歴史をもつ DB と OS の研究の成果が、とくに、トランザクション処理と仮想記憶についての成果が合流するところでもある。

最後に、MM-OODB の実現におけるプログラム言語の役割について少し触れておく。今日市場に出ている多くの OODB 製品では、ライブラリの形として OODB の機能が提供されている。コピー方式の GC ですら、言語に手をいれずに実現することは可能である。しかし、これではプログラマだけが負担を強いられることになる。プログラミングを容易にし、出来上がるプログラムを安全にするためには、コンパイラによる自動化や構文上のサポートが不可欠であると考える。

参考文献

[1] Hagmann, R.B. A Crash Recovery Scheme for a Memory-Resident Database System. *IEEE Transaction on Computers* (September 1986), 35(9)

[2] Birrell, A.D. et al. A Simple and Efficient Implementation for Small Databases. *Proceedings of the 11th Symposium on Operating System Principles* (1987).

[3] Wilson, P.R. el al. Effective "Static-Graph" Reorganization to Improve Locality in Garbage-Collected Systems. *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation* (1991).

データベース・システム 85－7－3
プログラミング－言語・基礎・実践－ 4－7－3
（1991. 10. 4）

永続的プログラミング言語と オブジェクト指向データベース

鶴岡 邦敏
日本電気㈱C＆Cシステム研究所
〒216 川崎市宮前区宮崎4－1－1（tsuruoka@btl.cl.nec.co.jp）

1．はじめに

次世代データベースとしてのオブジェクト指向データベースの研究開発が進展するにつれて、データベースとプログラミング言語の位置づけに関して、新たな枠組みが形成される機運がある。本稿では、オブジェクト指向データベース（OODB）を処理するために、永続的オブジェクトを扱うプログラミング言語（PPL）に必要とされる機能的拡張に関して述べる。これは、特定の言語の利用者から見てDB機能は透明であること、即ちその言語側のモデルとしてDB機能を持つことが必要、という立場による。特に、筆者らが開発中のオブジェクト指向DBMS－Odinの機能を踏まえて、OODB側から見たプログラミング言語への要請（＝拡張機能）を中心に記述する。OdinはC＋＋の拡張言語（Odin/C++）を持つため、以下ではC＋＋等の言語を想定する。

2．プログラミング言語への要請

（1）型とクラスの併用

以下では、同種のオブジェクトが持つ性質を型、その性質を定義したオブジェクトをクラスと考える。OODBにおいても、プログラムの誤動作を防ぎデータベースの完全性を確保する目的で、型（特に型検査）は重要である。一方以下の理由で、実行時に動くオブジェクトとして、クラスも必要である。

- クラスが保有する定義情報を実行時に参照し、それを解釈しながら動くユーティリティがある（例：ブラウザ、インタープリタ）。
- クラス変数やクラスメソッドを提供する。
- クラスの構造（型の定義体）を変更する際に、応用プログラムの修正や再コンパイルをできるだけ避ける（実行時にクラスを参照する）。

（2）ポインタ変数の回避

ポインタ変数（＊p等）は、プログラムの誤動作の原因となり易く（DBにおいてはデータ破壊を起こす危険性を持つ）、またプログラムの記述性を低下させる。このため、応用プログラムに対してはポインタ変数の不要な構文を提供すべきである。C＋＋系のOODBではポインタ変数で永続オブジェクトを指すものが多いが、以下のようにしてポインタ変数を回避すべきと考える。

- 基本型の変数はデータの値を持つ。
- クラス型の変数はオブジェクトへの参照を持つ。
- オブジェクトは他のオブジェクトの実体を含まない。

[Odin/C++の例]
```
社員  emp;
会社  comp;
comp = emp.hired;
```
[市販OODBの例]
```
persistent shain *emp;
persistent kaisha *comp;
comp = emp -> hired;
```

（3）型構成子と実行時に生成される型

OODBでは、DB中のオブジェクト群を部分的に見たい、構造を組み替えて見たい等の目的で、実行時に型を生成する場合がある。このため、型の名前の代わりに、型構成子を指定できることが必要である。

[Odin/C++の例]
```
set{社員}  emps;
set{会社}  comps;
select{社員}  selemp;
set{select{社員}}  selemps;
join{社員, 会社}  empcomp;
set{join{社員, 会社}}  empcomps;
selemps = emps.select(...);
empcomps = emps.join(...comps...);
```

（4）型構成子を持つクラスの定義

あるクラスを定義する場合に、その型を型構成子として与えたい場合がある。例えばOdinの集合クラスSimple_Setは、要素となるオブジェクトのクラスをクラス変数の値として持つ。この時Simple_Setのサブクラスを定義するためには、要素のクラスを指定する構文が必要となる。

[Odin/C++の例]
```
persistent class 社員集合
:public Simple_Set
:as set{社員}
{ };
```
上記の例では、:as 以下で要素の型を指定している。

（５）ビュークラス

多数の利用者が共有するＯＯＤＢにおいては、オブジェクト群の構造を利用者ごとに特有の観点から見るために、ビュー機能が必要とされる。Odinでは、結合、グループ化等の集合操作の結果として一時ビュークラスが生成され、また利用者が永続ビュークラスを定義することもできる。

```
[Odin/C++の例]
 persistent view class 社員会社
 :as join{社員, 会社}
 { projected:        // 可視変数、可視メソッド
    string empname;
    string compname;
   added:            // 付加メソッド
    ...
   public:
    meta void derive();  // ビュー導出メソッド
 };
```

永続ビュークラスにより、インスタンス変数やメソッドの参照の制限（可視条件）や、メソッドによる参照・更新の意味の記述（付加メソッド）等が実現できる。

（６）トランザクション構文

ＯＯＤＢにおいては、並行制御やデータベース処理の一貫性を維持する目的で、トランザクションの概念が必要とされる。あるトランザクション内で参照／更新されたオブジェクト群は、トランザクション終了時に発行されるcommitによって他のトランザクションとの競合をチェックされる。またrollbackによって、プログラマが意識的に更新を無効にすることも可能である。この際、トランザクション内のオブジェクトを外から参照できない、commit/rollbackを発行せずにトランザクションから抜けられない、等の制約を実現する構文が望まれる。このような目的で、トランザクション文が必要となる。

```
[Odin/C++の例]
 transaction Tr(...)
 { ...
  if(...) break_trans;
   ... }
```

（７）集合検索

大量のオブジェクト群が共有される環境においては、ある利用者が当面必要なオブジェクト群を特定するために、集合検索機能が重要である。

```
[Odin/C++の例]
 selemps = emps.select([self: E, $会社: C |
   where E.hired == C and C.compname == "NEC"]);
```

（８）その他の機能

マルチメディア処理等の応用を考慮すると、可変長変数／可変長配列が必要となる（動的な変数追加も要請される）。このため、コンパイル時にインスタンス変数値のオフセットが決定できない。

３．実現上の問題点と今後の方向

現在の言語処理系は、その実現において以下のような問題点がある。将来は、ＯＯＤＢとＰＰＬとが統合されると共に、利用者に一貫したプログラム開発環境を提供すべきである。

- 型（クラス）定義情報をソース形式でファイルに保管し、応用プログラムのコンパイル時にincludeして用いる。この方式だと、例えば対話型のユーティリティを用いてクラス定義の変更を行った場合など、ＤＢ中のクラス構造とinclude file中のクラス定義の不整合が生ずる恐れがある。型／クラス定義情報はオブジェクトとしてＤＢ中に格納し、これを言語処理系が直接参照すべきである。
- メモリ管理（ヒープ領域等）が大量のまたは長大オブジェクトの処理に適していない（ヒープが無制限に拡大する、実行中にメモリ割り当てが不可の状態に陥る等）。ＯＯＤＢ側のメモリ管理との統合を図り、仮想記憶を越える量のオブジェクトを処理でき、かつ安定したメモリ割り当てを行える管理方式を実現すべきである。
- エディタ、コンパイラ、デバッガ、ローダ等が開発環境として統合されていない。またこれらのツールは、「ファイル」という物理的な実体を意識した実現をとっている。将来はＯＯＤＢを中心として、以下のようにオブジェクトを管理する（ファイルを意識しない仮想的な）開発ツールとすべきである。

  エディタ：　「ソースメソッドクラス」のオブジェクトを編集するオブジェクトエディタ

  コンパイラ：　「ソースメソッドクラス」のオブジェクトを「実行形式メソッドクラス」のオブジェクトに変換するオブジェクトトランスレータ

  ローダ：　「実行形式メソッドクラス」のオブジェクトを動的リンクするダイナミックメソッドローダ

参考文献

[KiTs90]木村、鶴岡「ｵﾌﾞｼﾞｪｸﾄ指向ﾃﾞｰﾀﾍﾞｰｽ操作言語Odin/C++について」、電子情報通信学会データ工学研究会、DE90-26、1990.12.14

[KiTs91] Kimura, Y. and Tsuruoka, K. "A View Class Mechanism for Object-Oriented Database Systems," Proc. DASFAA'91, pp.269-273, April 1991.

データベース・システム 85－7－4
プログラミング－言語・基礎・実践－ 4－7－4
（1991. 10. 4）

データベースとプログラミング言語
－ソフトウェアパラダイム論からの考察－

布川博士
東北大学電気通信研究所

本稿で述べる内容は筆者自身の意見であり，広く一般にコンセンサスのあるものではない可能性があることをはじめに申し上げる．また，種々の概念の網羅的な解説でも，多くの意見を再整理したものでもないことをあらかじめお断りしておく．

1 はじめに

ソフトウェアのパラダイム論から見たとき，プログラミング言語やデータベースがどのように見えるかを議論する．

特に本稿で中心的に議論したい内容は，ソフトウェアシステムの実現手法の数々ではなくて，その計算モデルの特質と，より高階な考え方である，計算モデルを考えるためのソフトウェアパラダイム論である．

ソフトウェアパラダイム論の目的は，ソフトウェアを用いたよりすぐれた情報システムの構築である．この情報システムのためには，新たなソフトウェア構造，計算モデルが必要となる．

特に最近，ソフトウェア関連分野では，新たなカテゴリのソフトウェアシステムが登場している．この新しいソフトウェアカテゴリの探求・構築のためにも，メタセオリーとしてのソフトウェアのパラダイム論が非常に重要である．この状況下でプログラミング言語やデータベースがどのように構造化されて登場するかを議論する．

2 ソフトウェアシステム

情報システムは計算機とソフトウェアの融合体である．ソフトウェアシステムとは，このうち，ソフトウェアによって構成されるシステムである．ソフトウェアシステムは計算機システムや通信システムと異なり，ハードウェアによる制約がないため，ソフトウェアのみに焦点を当て，種々の側面から柔軟に，システム論，パラダイム論を議論できる特徴がある．

近年ソフトウェアシステムは大きく様変りしており，従来のキーワードでは語り切れないソフトウェアシステムが登場してきている．

たとえばハイパーカードのスタックウェアように，データの設定だけでソフトウェアになっているものもある．プログラミング言語を使わなくても，伝えたいことがらを容易に伝えてくれる．さらに，複雑な制御をしたいときのみ，HyperTalkというプログラミング言語を用いている．

はたして，ハイパーカードはデータベースか，ユーザインタフェースか，プログラム開発環境か？．ハイパーカードはハイパーカードという新しいパラダイムのソフトウェアシステムであり，ハイパーテキスト，ハイパーメディアにはない独自のカテゴリを形成している．同様にSmalltalk-80も単なる言語にとどまらない，優れたソフトウェアシステムの例である．

3 プログラミング言語

プログラミング言語は以下の２つの側面から議論されるべきである．

（1）計算モデルとして優れているかという側面（高級性）
（2）計算モデルは従来と同じでも，多くの機能を持っているかという側面（高機能性）

プログラミング言語の高級性の観点からの分類にはさまざまな切り口があるが，たとえば以下のようなものもあろう．

（1）構造化→抽象データ型→オブジェクト指向
（2）手続き型→非手続き型（関数型，論理型）
（3）並列・並行→分散型(distributed)→分散協調型(decentralized) [4]
（4）データパシング，メッセージパシング，RPC，．．．
（5）型付き，型無し，型階層，高階な型，型変数の有無
（6）その他，いろんな方法

ソフトウェアパラダイム論に基づくプログラミング言語の研究は，常に，この高級性の追及である．

プログラミング言語の高機能化は，実際にプログラミング言語を用いるためのアドホック的機能ともみるれが，工学的産物として実用化するために必要欠くべからざる機能である．

高機能性は，たとえば，特定の分野での応用のために，その分野専用の機能を導入することがある．データの永続化等はこの例である．

データの永続化に限ってみれば，属性リストをディスク上に書き込むことができるＬｉｓｐ[1]，抽象データ型の分散性を重視し，それを単位として永続性を持たせたArgus[2]などがある．Argusは障害発生からの自動復帰までもサポートしている．

4 データベースとプログラミング言語

データベースシステムの高級性はそのデータモデルの変遷で捉えることが出来る．すなわち

・ネットワーク，階層型→関係型→オブジェクト指向

また，データベースシステムの高機能化は，データモデルへ以下のような機能の追加によって行われる．

・データの蓄積機能，メタデータ管理機構，質問処理，トランザクション処理，．．．

この観点からオブジェクト指向データベースを見ると，オブジェクト指向データモデル（多くの面でオブジェクト指向計算モデルと共通しているところが多い）にデータベースのための高機能性を導入したものとみれる．

すなわち一般論としても，プログラミング言語と比べたときデータベースを特徴付けるのは，この高機能性の有無，優劣である．従来の永続的プログラミング言語はとりたててこの機能を重視していないものが多いように見受けられる．

データベースの門外漢としては，ここで述べたアプローチとは違った，ソフトウェアパラダイム論に基づた新たな高級性が，どのようにしてデータベースシステムに導入されてゆくかが非常に興味深いところである．

5 まとめ

新しいソフトウェアパラダイムを構築する際の中核として，プログラミング言語とデータベースはなくてはならない両輪である．そして，道路として分散環境が携わっていなければならない．

両輪は一般の乗物のように，左右別々にあるのではなく，たがいに融合された形で存在しなければならない．

ここで言う融合とは，(1)プログラミング言語の高級性とデータベースの高機能性の合成，(2)データベースの高級性とプログラミング言語の高機能性の合成．のいずれでもない．高級性や高機能性ではっきりと分けることの出来ない混合である．この混合されたソフトウェアシステムとは何かが非常に興味深い．そしてこのソフトウェアシステムは現実社会の高度なモデリングにとどまらない，ソフトウェアシステムだからこそできる新たな情報体系を提供するに違いない．

私個人としては，[3][4]の計算モデルに基づく[5]のシステムの構築を，データベースという観点から探求してみたいと思っている

謝辞

本稿は筆者が従来行ってきた研究をもとに，最近の研究課題[3][4][5][6]を通して得られた知見によるものである．研究を進めて行く上でご議論いただいた諸氏に感謝する．特にオブジェクト指向データベースに関しては，図書館情報大学増永良文教授との議論に負うところが大きい．本稿に関しても多大なコメントを頂戴した．感謝する．

参考文献

[1] 電総研：Petlマニュアル(1982)

[2] Liskov, B. and Scheifer, R. : Gurdians and Actions : Linguistic Support for Robust Distributed Programs, ACM Trans on Prog. Lang. and Syst. Vol.5, No.3(1983)

[3] 矢野，布川，野口：メッセンジャー伝播による分散プログラミング，ソフトウェア科学会第８会大会講演論文集(1991)

[4] 矢野，武宮，布川，野口：自律的な協調を行う分権型計算モデルKemari，情処研報，Vol.90-PL27(1990)，pp.151-158

[5] 布川，野口：メタファーネットワークWhy and How，第７回ヒューマンインタフェースシンポジウム講演論文集（1991）

[6] 福田，村田，吉村，村田，布川，増永：ネットワーク環境のＯＯＤＢを用いたデータモデリングの試み，情処研報，データベースシステム研究会(1991年，１１月，掲載予定)

[7] 増永：次世代データベースシステムとしてのオブジェクト指向データベースシステム，情報処理学会誌，Vol. 32, No. 5 (1991), pp.490-499

データベース・システム 85－7－5
プログラミング－言語・基礎・実践－ 4－7－5
（1991. 10. 4）

永続プログラミング言語研究開発の勧め

牧之内顕文
九州大学工学部情報工学科

昔、ＩＢＭがまだ巨人としてコンピュータ業界に君臨していた頃、私は日本のとあるコンピュータメーカに入社した。新人教育の一環として、ＯＳに関する一連の講義を受講した。ジョブ、タスク、サブタスク等々の難しい概念に閉口させられた覚えがある。その一連の講義の講師の一人が、「データ管理（いわゆる「デ管」）については難しいので（即ち、新人諸君に話しても諸君は理解出来ないだろうから）省略する」と言ったことを今でもハッキリ思い出す。その後、何の因果か、データベースシステムの研究開発をするようになり、やむを得ず（メインフレームの）データ管理システム（データ編成方、アクセス法）を勉強した。

1970年代初頭、プログラミング言語の設計やそのコンパイラ開発の仕事に携わったことがある。その時には、プログラミング言語のデータ（変数）に関する種々の概念 － 変数、定数、静的(static)データ、動的(dynamic)データ、自動(automatic)データ、グローバル、ローカル、スコープ、レジスタ、スタック等々 － に悩まされた。しかし、プログラミング言語に於けるデータに関する複雑な諸概念の存在にもかかわらず、ファイルデータに関する扱いは（順編成ファイル）のREAD、WRITE で片づけられていたのは不思議であった。この理由としては、データ管理が複雑すぎてプログラミング言語設計者にも理解できないのであろうと思ったりもした。ちなみに、Algol 60 がその後のプログラミング言語設計に及ぼした影響は計り知れないのに、使われず、従って普及しなかったのは「I/O 」が無かったからだと言われている。

以来、プログラミング言語についての研究は色々行われてきたがどうも私には「新鮮さ」がないように感じられた。ただ、Prologの出現はプログラムの実行制御に関して新しい視点を拓いたものと評価される。しかし、この言語でもデータの生存期間については従来型の言語と同様な扱いである。そこで、私は永続プログラミング言語の研究をプログラミング言語研究者の方々にお勧めしたい。この主の言語は従来の言語の多様な研究テーマに新しい刺激を与えるものと信じる。例えば、

（1）データのスコープ論
（2）データタイプ論
（3）オペレーション論
（4）最適化論

等々

さらに、従来のプログラミング言語論には存在していない概念 － トランザクション、データ復旧、機密保護など － を新たに咀嚼・統合しなくてはならない。あるいは、データベースでは必須のデータの競合管理はプログラミング言語ではまだ表面に出ていないようだが並列プログラミング言語では必須の機能になるはずである。このための「ロック」にかんする研究はデータベース分野では精力的に行われたが、その成果は永続プログラミング言語研究では生かされるものと信じる。