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

ディクショナリをベースにしたNATURALによる第4世代アプリケーション開発

吉舗紀子

(株)ソフトウェア・エージー

COBOL等の第3世代言語を用いたアプリケーション開発に対し、データ・ディクショナリと統合化された第4世代アプリケーション開発ツールを用いた開発方法について、その基本的考え方と特徴を述べる。例として、NATURALとPREDICTを取りあげ、これらの機能の概要とこれらを用いた開発方法を紹介する。さらに、アプリケーション保守やリエンジニアリングへの対応についても言及する。

Fourth Generation Application Development using NATURAL based on a Data Dictionary

Noriko Yoshiki
Software AG of Far East Inc.

Shinjuku L Tower, 7F. 1 - 6 -1, Nishi-Shinjuku Shinjuku-ku, Tokyo 160, Japan

This paper describes basic concepts and features of application development using fourth generation tools integrated with a data dictionary in comparison with application development using a third generation language such as COBOL. It introduces development methodology using NATURAL and PREDICT as an example giving a summary of their functionality. Moreover, it refers to application maintenance and re-engineering.

1. はじめに

あらゆる企業･組織において情報の重要性はいうまでもないが、コンピュータをベースにした情報システム構築の成否がビジネス成功の鍵となってきており、ソフトウェアに対して多額の投資が行われている。

情報システムの開発のみならず、組織を取り巻く環境の変化に対応してシステム保守を適時に行っていくことが重要課題となってきている。このようにシステムの保守をもタイムリーにかつ適切に行っていくためには、データに関する情報の他、アプリケーション･プログラムそのものに関する情報やプログラムとデータとの関連情報もコンピュータにより管理する必要があり、保守まで含めたシステムのライフサイクル全体を管理するディクショナリ･システムと、このディクショナリと完全に統合化された開発ツールが必要である。

2. 第4世代テクノロジとオブジェクト指向アプローチ

第4世代テクノロジとは、COBOL等の第3世代言語を用いたアプリケーション開発に対し、単なるプログラム作成言語としての第4世代言語のみでなく、アプリケーション開発に必要な機能を備えた統合化ツールによる開発を意味する。この第4世代テクノロジ自身がすでにオブジェクト指向であるといえる。それは次の理由による。

アプリケーションの構築については、構造化設計により機能をブロックに分け、それらをもとに組立る方式が望ましい。第4世代テクノロジでは、論理データビューの他、データ検証のための処理ルール等をディクショナリにオブジェクトとして保持し、さらにアプリケーション内のユーザ･インタフェース部分、すなわち、画面マップ、ヘルプ(ユーザ･ナビゲーション)、レポート･フォーム等もあらかじめオブジェクトとして定義できる。

したがって、アプリケーションはCOBOLを用いて数千行もコーディングするのではなく、第4世代アプリケーション開発環境の中では、論理的な完全性がチェックされた定義オブジェクトを組み合せることによりインプリメントする。したがって、ミスが少なくなり、アプリケーションの品質が向上し、保守も容易になる。

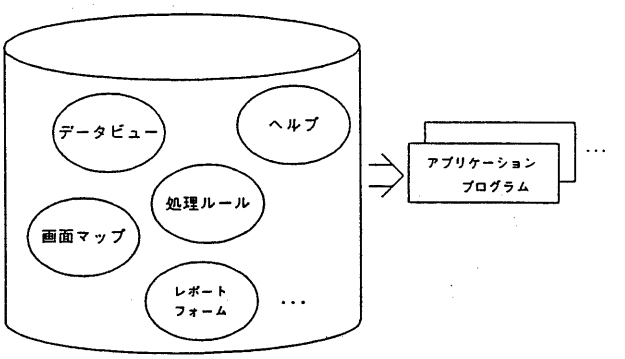

図１　第4世代テクノロジによる
アプリケーション開発

以降は、ソフトウェア･エージー社の第4世代アプリケーション開発ツールNATURALとアクティブ統合ディクショナリPREDICTを例として、第4世代アプリケーション開発方法を記述する。

3. データ･ディクショナリの必要性とPREDICT

(1) データ･ディクショナリの要件

データ･ディクショナリは、データおよびアプリケーション･システムに関する情報、しかも活きた情報を保持するセントラル･リポジトリとして働き、以下の要件を満足するものでなければならない。

- アプリケーション･システムに関する情報がシステム完成後の変更も含め一貫性を保っていること

- アプリケーション・システムに関するドキュメント作成に必要な労力を軽減するものであること

したがって、手入力で情報をディクショナリに登録するのではなく、自動的に情報を採取し(自動ドキュメンテーション)、また必要に応じてディクショナリ内の情報をアプリケーション・プログラムにも自動適用する仕組みがなければならない。

(2) PREDICTの機能と役割

PREDICTは情報資源管理、ソフトウェア管理、ドキュメンテーション支援のためのアクティブ・ディクショナリ・システムであり、NATURAL、PREDICT CASE(要求分析・設計支援ツール)、NATURAL CONSTRUCT(アプリケーション・ジェネレータ)を含むCASEテクノロジの中核を成すものである。図2にPREDICTの概要を示す。

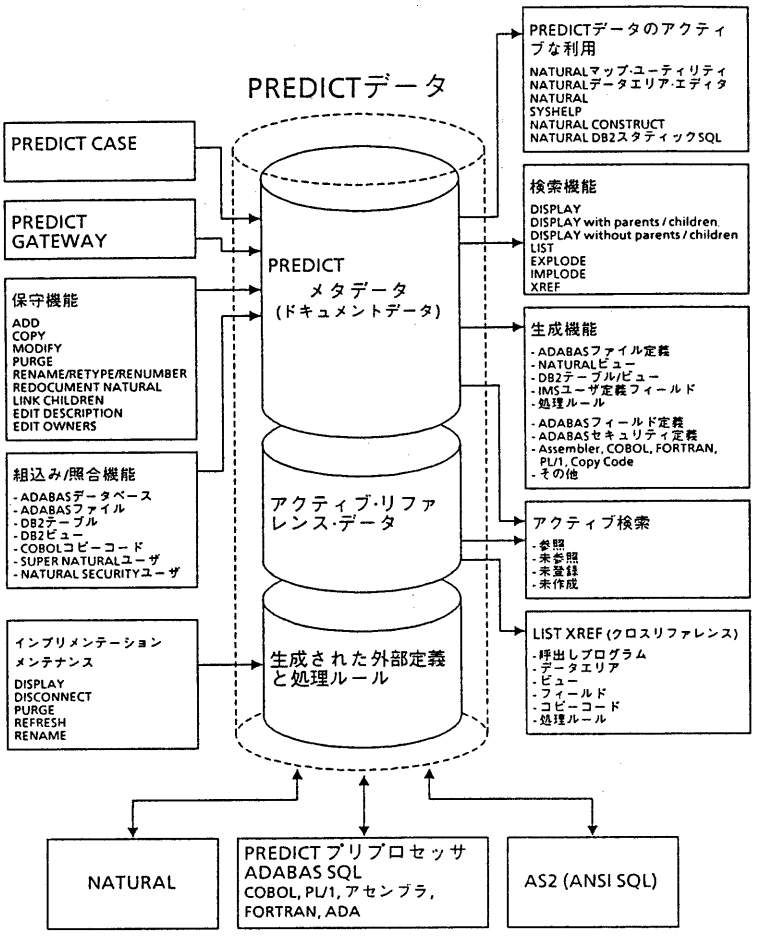

図 2.　PREDICTの概要

データベース設計については、入力伝票、出力帳票などシステムで扱う情報をまず「概念ファイル」としてPREDICTディクショナリに登録し、その後項目の重複等を除去し、項目辞書としての「スタンダード・ファイル」を作成できる。この後、正規化、物理設計等の作業を行い各々のファイルに切り出し、物理ファイル、ユーザビュー情報等を作成する。項目に対する変更は、スタンダード・ファイルに対して行えば、そこからひき出された定義情報はすべて自動更新される。

さらにソフトウェア・ライフサイクルの各段階において、PREDICTは以下のような役割を果たす。

設計段階

- アプリケーションのデータモデル(ビュー)の定義
- アプリケーションの機能設計の概略定義
- プロジェクト・チームの構成とチーム・メンバへのタスクの割当て

開発段階

- アプリケーション開発に必要な情報の提供
- データ定義、コピーコード等の生成
- PREDICT内の記述情報からヘルプ･テキストを生成
- データの一貫性の保証(データ定義、データ･チェック･ルールの適用)
- アクティブ･リファレンス･データにより開発の進捗状況の制御
- アプリケーションのバージョン管理(PREDICT APPLICATION CONTROL)

検証/テスト段階

- テストプラン作成に必要な情報の検索
- アクティブ･リファレンス･データを用いて開発の一貫性を保証

保守段階

- 変更に伴う影響の範囲を予測
- アプリケーション変更時におけるデータ定義、機能構造の一貫性を保証
- 保守の対象となる該当するすべての要素の検索
- 保守作業の立案を支援

上記の他、ライフサイクルのすべての段階においてPREDICTの検索機能を使用し、アプリケーションに関する情報を検索したり、必要に応じてカスタマイズしレポートを作成することができる。

4. 第4世代言語とNATURAL

(1) NATURALの開発思想

NATURALは、ユーザ指向のまったく新しいアプリケーション開発環境として1979年に誕生した。「第4世代言語」という言葉が登場するより前にNATURAL自身はすでに存在していた。

プログラミング言語はJ. Martin氏によれば、以下のような分類になる。[1]

第1世代言語	機械語
第2世代言語	アセンブラ言語
第3世代言語	COBOL, PL/1, FORTRANなど
第4世代言語	NATURAL, FOCUS, MANTISなど

また、第4世代言語とは以下の特徴を持っているとしている。

① ユーザ･フレンドリである。

② 専門家でないプログラマでも結果を得られる。

③ データベース管理システムを直接使用する。

④ 手続型の場合はCOBOLの10分の1の命令数で結果が得られる。

⑤ できる限り非手続型コードを使用している。

⑥ レポート形式の選択などを自動的に行うことができる。

⑦ オンライン･オペレーション用として設計されている。

⑧ 構造化プログラミングが容易にできる。

⑨ 他人の書いたプログラムを容易に理解し、修正できる。

⑩ 一般的ユーザが2日間のコースで言語のサブセットを修得できる。

⑪ 容易にデバック出来るように設計されている。

⑫ COBOLやPL/1の場合の10分の1以下の時間で結果が得られる。

NATURALは、上記を満足すると共に以下の要件を満たすソフトウェアとして開発されたものである。

- ハードウェア、システム・ソフトウェアから完全に独立した形で機能し、自由にハード/ソフトを選択できること。
- 異なるデータベース環境でもトランスペアレントに操作できること。
- 分散環境であっても意識することなくプログラミングできること。
- 完全に会話型でアプリケーションの作成、デバッグ、テストが一連の流れとして行えること。
- プロトタイピング方式の開発に適合する画面作成などの機能が充実していること。
- すべてのデータ項目の情報を管理し、プログラム・ドキュメンテーションを自動的に作成するアクティブ・ディクショナリと統合化されていること。
- マップ、ビュー、データエリア、処理ロジックなどのコンポーネントを扱うためのインテリジェント・エディタとメニュー・ドリブンのアクティブ・ワークステーション機能。
- 簡単なデータ操作と柔軟なデータ処理を行う非手続型の言語機能。
- 大規模かつ複雑なアプリケーション作成のための広範な機能。
- アプリケーションの詳細な分析とプログラム構造の定義のためのコンピュータ支援ソフトウェア・エンジニアリング(CASE)機能と統合化されていること。
- ディクショナリと統合化されたプログラム、マップなどのライブラリ管理およびデータとプログラム環境を統合するセキュリティ機能。

NATURALは、このように「言語」としての機能の他、アプリケーション開発に必要な機能が1つの統合化されたシステムとしてまとめられた「第4世代アプリケーション開発システム」として位置づけられる。

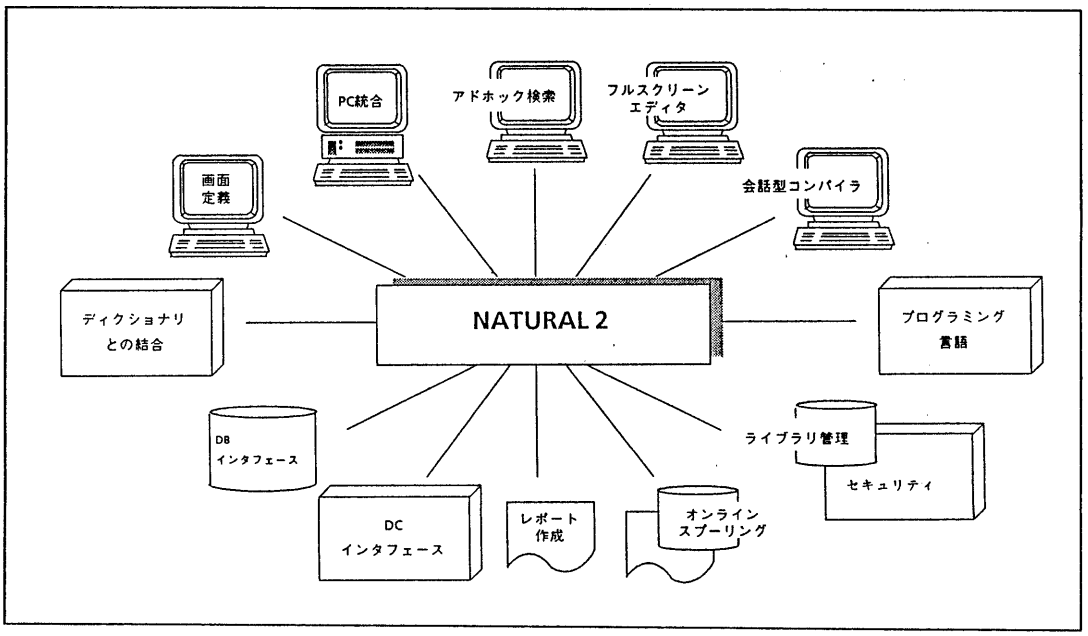

図3　統合開発システムとしてのNATURAL

(2) プログラム構成要素とディクショナリとの対応

NATURALによるアプリケーション･プログラムの構成要素は以下のとおりである。

① データエリア
- グローバル･データエリア … プログラム間共通のシステム共通エリア。
- ローカル･データエリア … あるプログラム内のみ有効なエリア。
- パラメータ･データエリア … プログラムとサブプログラム間のデータ受渡しのためのエリア。

② サブプログラム、サブルーチン
共通処理を行うもので他プログラムから呼び出されるもの。

③ 画面マップ
画面イメージで定義された画面レイアウト。画面自身や画面上の項目に⑤の処理ルール、⑥のヘルプ･ルーチンを割当て可能。複数プログラムで共通使用可能。

④ レポート･フォーム
レポート･イメージで定義されたレポート･レイアウト。複数プログラムで共通使用可能。

⑤ 処理(チェック)ルール
入力項目のチェック処理や画面にまつわる処理など。

⑥ ヘルプ･ルーチン
画面上にヘルプ情報を表示するユーザ･ナビゲーションのためのルーチン。

これらの相互関係、NATURALプログラムとPREDICTディクショナリとの関連を図4に示す。ディクショナリに登録されたビュー情報や処理(チェック)ルールはNATURALプログラムに自動適用される。

また、プログラムと使用データ項目、読み込みか更新か
プログラムと使用マップ
プログラムと使用レポート･フォーム
プログラムと使用サブルーチン、サブプログラム
プログラムと使用コピーコード

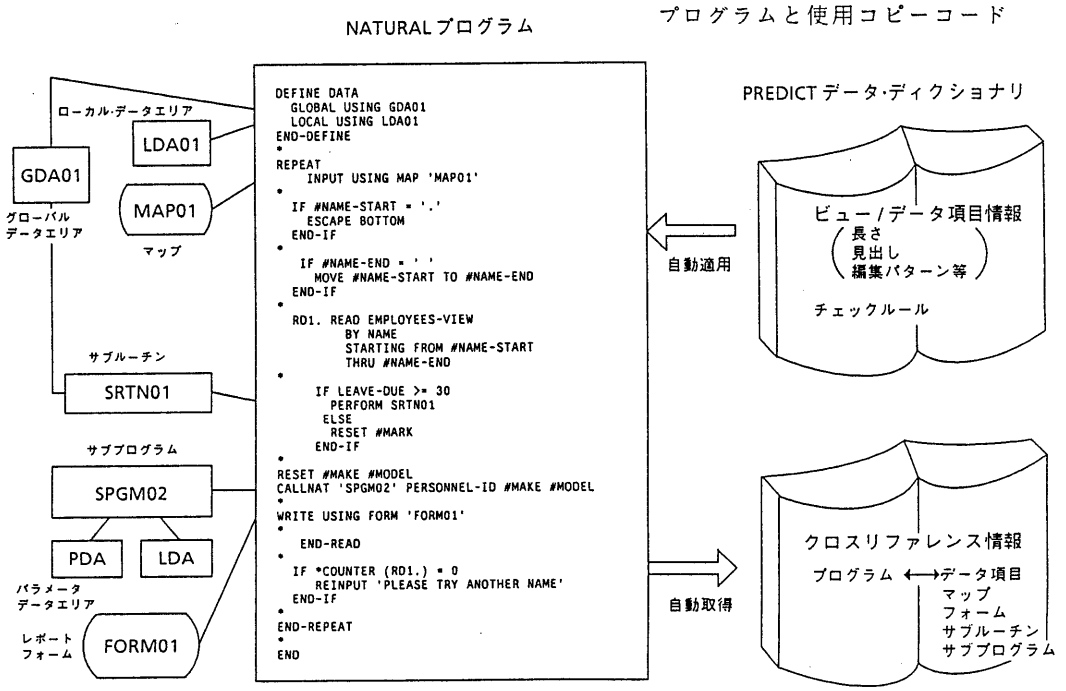

図 4. NATURAL プログラムとディクショナリ

という、クロスリファレンス情報が採取され自動的にディクショナリに書き込まれる。

したがって、変更に伴う影響の範囲を容易に入手することができ、保守の対象を確実に把握し、この情報に基づいて適切な保守作業を行える。

5. NATURAL, PREDICTによる第4世代アプリケーション開発と共通仕様

(1) アプリケーションの位置づけと構成部分

アプリケーション・システムとは、一方が「ユーザ」、他方が「データ」という両者の間に位置するものといえる。したがって、アプリケーションは次のような部分(レベル)に分けて考えることができる。

- ユーザとコミュニケートする部分(ユーザ・インタフェース)
- 実際のアプリケーション・ロジック
- 論理データビューおよび物理データビューなどのデータ記述

アプリケーションをこのようにとらえ、個々のレベルを明確に区別することは、環境変化への依存度をより少なくするという利点がある。例えば、ユーザ・インタフェース部分がユーザ要求を満たさなければ、他のレベルへ影響を与えることなく変更し置き換えることができる。

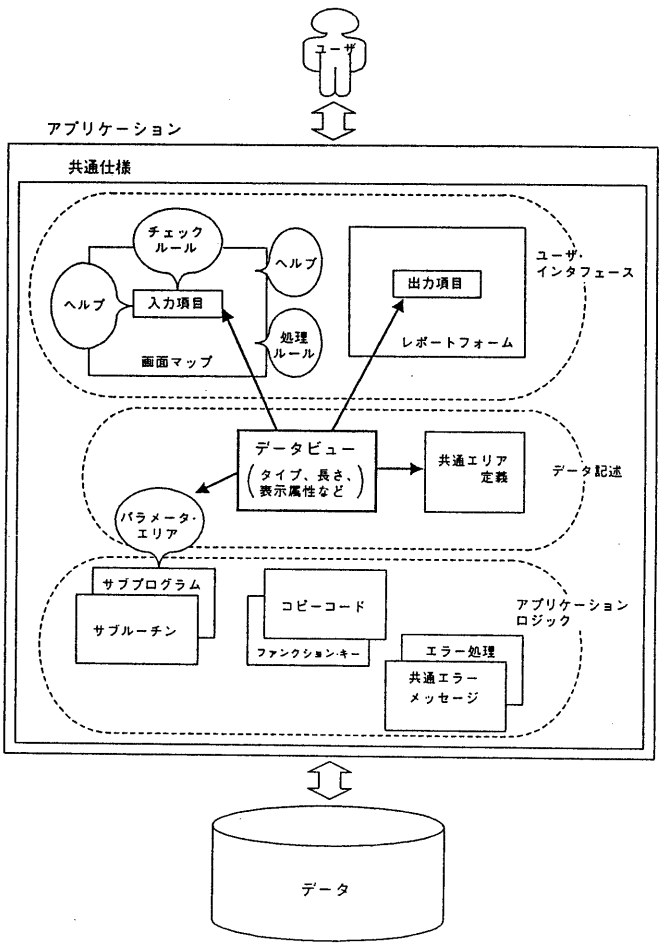

図5. 共通仕様とアプリケーション

(2) 共通仕様

NATURAL、PREDICTによるアプリケーション開発ではオブジェクト指向の考え方を取り入れ、アプリケーションを構成する上記各レベルについて、共通機能部分に構造化し、それをオブジェクトすなわち「共通仕様」として設計段階で作成する。

これにより、アプリケーションは、これらの組合せに個々のアプリケーションに応じたユーザ・コーディングを付加して構築することができる。

開発ツールとしてNATURAL、ディクショナリとしてPREDICTを使用した場合に設定できるアプリケーションに関する共通仕様を図5に示す。ディクショナリに定義されたデータ項目自身の属性や表示属性(見出し、編集指示、長さ)は画面の入力項目やレポートの

出力項目に対して自動適用される。また、関連するチェックルールやヘルプを規定しておけばアプリケーションに自動適用される。

ユーザ･インタフェース部分は、外部マップ、ヘルプ機能(ユーザ･ナビゲート)、入力チェック、レポートフォームから構成される。このレベルでは、エンドユーザと共同でプロトタイピング方式で共通仕様を作成する。

アプリケーション･ロジック部分の共通仕様は、サブルーチン、サブプログラム、コピーコード、エラー処理等である。ファンクション･キー設定の共通化も行える。エラーに対するメッセージ、共通の実行時のエラー処理も設定できる。共通メッセージに関しては、メッセージ番号と対応メッセージを登録しておけば、メッセージ番号を指定して取り出すことができ、処理(チェック)ルール内での使用も可能である。

データ記述部分は、グローバル･データエリアやパラメータ･データエリアなどの共通エリア定義である。データ項目に関してはビュー名で参照するだけであり、項目属性はディクショナリから自動適用される。

6. 第4世代テクノロジによるアプリケーション開発の特徴

ディクショナリをベースにした第4世代テクノロジによるアプリケーション開発の特徴は、次のようにまとめることができる。

- プロトタイピングによるユーザ･インタフェース設計
  会話型でイメージどおりに画面やレポート･レイアウトを作成し、その成果物がそのまま設計ドキュメントとして、またアプリケーションの構成要素として使用される。入力チェックやヘルプもこの段階で作成･検証でき、エンドユーザと共に設計できる。
- 共通機能設計によるユーザ･コーディングの軽減
  共通処理ロジックとしてのサブルーチン、サブプログラムの他、ファンクション･キーの共通設定、共通エラー処理、エラーメッセージなどの共通機能をコピーコードと併せ設定でき、生産性向上と共にでき上がったシステムの品質が保証される。
- 開発の進捗管理と一貫性の保証
  PREDICTのアクティブ･リファレンス機能および照合機能により、開発の進捗状況をモニタしたり、設計情報と開発情報の相違をみつけ、適切な修正を施すことができる。

7. 保守支援とリエンジニアリングへの対応

既存のアプリケーションの変更や拡張は、一貫性のあるドキュメントが存在していることが前提条件である。また、既存のアプリケーションからドキュメントを生成できるリエンジニアリング機能も重要である。PREDICT、NATURALによる第4世代テクノロジでは、変更に伴う影響分析を行える他、組込み(INCORPORATE)機能により既存のアプリケーション構成要素からドキュメントを自動作成できる。また、REDOCUMENT機能により、既存プログラムに関して、クロスリファレンス情報や更新ステートメント使用有無、コメント情報等をディクショナリに取り込むこともできる。

このように、常にアクティブな一貫性のある情報をディクショナリに保持し、このディクショナリと統合化された第4世代開発ツールを用いることにより、単にアプリケーション開発を早期に完了させるだけではなく保守までも包含した本当の意味での生産性の向上を実現することができる。

参考文献

[1]James Martin, 4GL Vol.Ⅱ, Prentice-Hall, 1986

[2]石井義興「第4世代言語について」私大協大会講演予稿集、1987年 9月

[3]末舛史郎、吉舗紀子「第4世代言語NATURALを用いたプロトタイピング方式のシステム開発事例」情報処理学会、1986年 4月

[4]ビル·ワグナー(テキサス大学)「第4世代への懸念 - よくいわれることだが根拠がない」Computer Report、1989年 6月号

[5]ソフトウェア·エージー「PREDICT Concepts and Facilities」