

パネル討論:

次世代データベースシステム宣言に向けて
—ソフトウェア・オブジェクトベース構築の立場から—

司会 田中克己(神戸大)

パネリスト 増永良文(情報大)、田中克己(神戸大)、片山卓也(東工大)、
所真理雄(慶大)、徳田雄洋(東工大)、デビッドノトキン(ワシントン大)

近年、オブジェクト指向データベースシステムはその強力なモデリング能力やプログラミング言語との融合性の高さから多くの注目を集めている。さらに、昨今提唱されたOODB宣言や、これに対する第3世代DBMS宣言など、いわゆる次世代データベースシステムに関するmanifestoが発表され、次世代データベースシステム像に関する議論が国内外で活発化している。本パネル討論は、本年7月のデータベースシステム研究会でのパネル討論に続く2回目で、我が国における情報処理関連の研究コミュニティにおいて期待される次世代データベースシステム像を、CASEデータベースなどのいわゆる「ソフトウェアオブジェクトベース」を構築する立場から討論を行なうものである。

Panel Discussion:

Towards a Next-Generation Database System Manifesto
- A View from Constructing Software Object Bases -

Chaired by: Katsumi TANAKA (Kobe U.)

Panelists: Yoshifumi MASUNAGA (U. of Library & Information Science),
Katsumi TANAKA (Kobe U.),
Takuya KATAYAMA (Tokyo Institute of Technology),
Mario TOKORO (Keio U.),
Takehiro TOKUDA (Tokyo Institute of Technology),
David NOTKIN (Washington U.)

Recently, much attention has been focussed on Object-Oriented Database Systems (OODBMS), especially for their strong modelling capability and the good interface to programming languages. *The OODB manifesto* and its responder: *the 3rd Generation DBMS manifesto* triggered world-wide intensive discussions about the desirable features of next-generation database systems.

This panel discussion is the 2nd one, which follows the 1st panel discussion held at July 1990 by the SIGDBS research meeting, and this time, we will discuss the desirable features of next-generation database systems, especially from the viewpoint of constructing *Software Object Bases* such as CASE databases.

次世代データベースシステム宣言について

神戸大学工学部・田中克己

On Next-Generation Data Base System Manifesto Katsumi Tanaka (Faculty of Engineering, Kobe University)

1. オブジェクト指向データベースシステム宣言と第3世代データベースシステム宣言

オブジェクト指向データベースシステム宣言[1]は、オブジェクト指向DBMSが満たすべき条件を、必須条件、付加的条件、選択条件に分類してまとめ上げたもので、これを以下に要約する。

必須条件

(1)複合オブジェクト、(2)オブジェクト識別性、(3)カプセル化、(4)型／クラス、(5)型階層／クラス階層と継承、(6)メソッドの再定義、(7)メソッドの多重定義、(8)遅延束縛、(9)計算の完全性、(10)拡張可能性、(11)オブジェクトの永続性、(12)2次記憶管理、(13)並行処理制御、(14)障害回復、(15)問い合わせ処理機構

付加的条件

(1)多重継承、(2)型検査と型推論、(3)分散、(4)設計トランザクション、(5)バージョン

選択条件

(1)プログラミングパラダイム、(2)表現システム、(3)型システム、(4)統一性

第3世代データベースシステム宣言[2]は、レコード集合に対する統一的なデータ定義言語／データ操作言語というDBMSの本質的な機能を提供した階層型・CODASYL型データベースシステムを第1世代、非手続きのデータ操作言語と高度なデータ独立性を提供した関係データベースシステムを第2世代のデータベースシステムとして位置づけ、さらに、これを踏まえて、次に来るべき第3世代のデータベースシステムが具備すべき条件を提案している(以下参照)。これらの条件を満たす次世代DBMSは、現状のオブジェクト指向DBMSでは不十分であり、それよりも既存の関係型DBMSの拡張というアプローチで達成できるという立場をとっている。

(1)豊富なオブジェクト構造とルールを扱う機能

- (a)型システム(抽象データ型、型構成子、複合オブジェクト)のサポート
- (b)(多重)継承機能
- (c)データベース操作の関数定義機能とカプセル化機能
- (d)プライマリキーの代替としてのオブジェクト識別子のサポート
- (e)トリガーや完全性制約を実現するための独立なルール管理機能

(2)第2世代DBMSの機能を包含できること。

- (a)非手続きの／高水準問合せ機能によるプログラムからのDBアクセスとDBMSによる最適化
- (b)集合の外延的記述／内包的記述機能(属性値としてqueryを許す機能)
- (c)更新可能なビュー機能
- (d)データモデルの効率向上のための機能からの独立性

(3)他システムからの開放性

- (a)複数の高水準言語からのアクセス機能(Multi-lingual DBMS)
- (b)永続的プログラミング言語のサポート
- (c)SQLのサポート
- (d)問合せ(query)／検索結果のデータ集合がクライアント・サーバー間の転送単位。

2. 議論

ここでは、従来のDBMSが提供してきたいくつかの基本概念(データモデル、データ定義/操作言語、3層スキーマとデータ独立性、完全性など)、および、オブジェクト指向の基本概念を考慮しつつ、次世代データベースとしてオブジェクト指向DBMSがサポートすべき諸概念/機能について述べる。

2.1 従来のスキーマ/インスタンスの概念の修正

情報の基本単位はオブジェクトであり、従来のように、スキーマにインスタンス集合が対応するのではなく、(複合)オブジェクトの構成子(型構成子)が直交し、互いに、対等であるべきである。すなわち、従来のスキーマはここではオブジェクト名と解釈されるべきである。さらに、従来のスキーマとは異なり、メソッドをも含んだスキーマの概念が必要である。オブジェクト名としてのスキーマをもオブジェクトとできる機能については議論を要するが、あれば有用である。

2.2 関連型(relationship types)構成子の拡張性

関連型は必要最低限のものにとどめ、必要な場合には利用者が定義(再定義、多重定義)できるような拡張性を持つべきである。システム自身が固定化された関連型集合を提供することは、意味データモデルと同様の方向に陥る可能性がある。

2.3 情報隠蔽

複雑巨大なオブジェクト構造を、データベース操作を行なうという名目だけで、情報隠蔽をやぶって応用プログラム側に見せるべきではないと考える。これは、単なる利用者ビューの単純化ということだけにとどまらず、複合オブジェクトのデータ構造自身の進化(evolution)が十分予想されるためである。これに対して従来の非正規関係データベースなどでは解決に有用な手段を提供していないことに留意すべきである。なお、データベーススキーマとしては、複合オブジェクト(属性値が再びオブジェクト)という概念で十分であり、属性値がオブジェクトへのポインタなのかオブジェクト自身なのかの区別は、基本的には見えない方がよいと考える(ポインタ透明性)。

2.4 オブジェクト操作言語

オブジェクトの定義/検索/更新/その他の操作を行なうための、adhocな言語の必要性は高く、これは、必要ならばデータベースプログラミング言語と併用すべきものとする。このようなオブジェクト操作言語は、既存の集合論的問い合わせ言語ではなく、複合オブジェクト内の系統的なnavigation、繰り返し操作をサポートすべきであり、単なる集合操作ではおさまらない点に留意すべきである。また、この言語のレベルでは上述のポインタ透明性もサポートすべきである。既存のOODBMSはデータベース中に存在するオブジェクトしか検索できないが、これでは不十分であり、新しいオブジェクトの生成/合成する機能も必要である。

2.5 3層スキーマ

具体的なデータ構造やメソッド実現などを表現する「型」というものと、データベース利用者からみえる概念的なクラスは区別できると有用である(型/概念的クラスの区別)。これは、型の異なるオブジェクトの集合が1つのインスタンスになることが十分ありえるからである。すなわち、ここでの型は、従来のDBMSの3層スキーマにおける「内部スキーマinternal schema」としてとらえる方がよいかと考えられる。具体的なデータ構造やメソッド実現は、変更の可能性があることに留意すべきである。また、概念的クラスに対してはさまざまなviewを提供することが、論理的データ独立性/semantic relativismを実現する上で、必須であると考ええる。

参考文献

- [1] Atkinson, M. et al., *The Object-Oriented Database System Manifesto*, Proc. of the 1st International Conference on Deductive and Object-Oriented Databases (DOOD'89), pp.40-57, Dec.1989.
- [2] The Committee for Advanced DBMS Function, *Third-Generation Data Base System Manifesto*, Memorandum No.UCB/ERL M90/28, April 1990.

ソフトウェア開発のためのオブジェクトベース

東京工業大学情報工学科 片山卓也

1. はじめに

社会の高度化にともない大規模かつ複雑なソフトウェアを信頼性高く作成する技術が強く求められている。ソフトウェアの構造の数理的研究、ソフトウェア作成方法論や管理技法の進歩などによってこのようなソフトウェアを効率良く作る技術を我々は少しづつではあるが手にいれようとしている。それと同時にソフトウェアを作成する環境の重要性が認識され、ソフトウェアプロセスやそれから発生する生成物を格納するためのソフトウェアオブジェクトベースの研究が盛んになってきている。新しいデータベースに対する要求が第3世代データベース宣言などで指摘されているが、本稿ではソフトウェア開発のためのオブジェクトベースと関連して、これからのデータベースに対する問題提起を行なってみたい。

2. 多様なオブジェクト / ill-defined なオブジェクト

ソフトウェアオブジェクトベースは、ソフトウェア環境の中心に位置しており、その中で生成される全てのオブジェクトを格納したり、検索することが要求される。そのようなオブジェクトとしては、仕様書、設計書、ソースコード、オブジェクトコード、テストデータやQ Aデータなどのような文書やいわゆるデータ類のほかに、プロセススクリプトやプロジェクト管理情報などの多様なデータが格納される。これらのオブジェクトは、その形や大きさがまちまちであり、従来の定型的データの効率的格納や検索を目的としたビジネスデータベースでは対応が困難である。

ソフトウェアオブジェクトベースの特長は、単に多様な形のデータを扱わなければならないのみでなく、不明確なデータを扱わなければならない点である。ソフトウェアの作成、設計過程を記録し、その分析や再利用をととしてソフトウェアの構成プロセスを向上させようというソフトウェアプロセスの研究が活発に行なわれているが、そのようなプロセスの記録には柔軟な形のデータベースが必要である。また、メモやアイディアなどを組織的に管理することもソフトウェア設計の過程では重要である。このような目的ためにハイパーテキストの応用が試みられているが、ill-defined なデータのためのオブジェクトベースについての研究をもっと活発に行なう必要があろう。

3. オブジェクト間の関係記述とその維持

ソフトウェアオブジェクトベースでは、各オブジェクトは単独で存在する事はまれであり、多くの場合オブジェクトはお互いに色々な関係を満たしているのが普通である。このような関係としては、部品関係、バージョン関係、依存／導出関係、定義／使用関係、さらには一般化／詳細化関係など種

々のものがある。ソフトウェアは、通常多数（場合によっては数百から数千）の部品から構成されるのが普通であり、また、それらの部品にはその完成の度合いや仕様の違いによって多くのバージョンがあり、それらのなかから所定の物を選択して一つのソフトウェアを構成するための構成管理／バージョン管理機能はソフトウェアオブジェクトベースにとって最も基本的である。

あるオブジェクトが変化した場合、それと関係のあるオブジェクトはオブジェクト間の関係が満足されるように変化し、オブジェクトベース全体を矛盾のないようにすることが要求される。これは、通常は自動的に行なわれるが、場合によっては、試験的に矛盾のある状態を作りだしたり、あるいは、オブジェクトに対するアクセスが発生するまで変化の伝播を遅らせるなどの方策がとられることがある。

4. 進化／変化

ソフトウェアの最大の特長が変化 (change) および進化 (evolution) にあることは広く認識されている。実社会に組み込まれて使用されるソフトウェアは社会の変化に対応して変化／進化する事によってのみその寿命を延ばすことが出来る。また、人間の作りだすソフトウェアに基本的には未完成であったり誤りがつきものである。完成度を高める過程でもオブジェクトとしてのソフトウェアは変化しなければならない。さらには、ソフトウェアの作成過程にも変化を要求するファクターがある。例えば、ソフトウェアは種々のツールによって作られるが、ツール自身進化しており、それに対応してオブジェクトを変化させる必要が生じる。

現在、我々はこのようなソフトウェア進化のメカニズムを手にいれておらず、これがソフトウェア保守コストの増大を招いており、また、保守不能なソフトウェアによって大きな社会問題が引き起こされる可能性もある。進化可能ソフトウェアの構成法はソフトウェア工学最大の研究テーマであるが、ソフトウェアオブジェクトベースにも進化や変化を可能にするメカニズムが備わっていない。前項で述べたオブジェクト間の関係の維持機構やバージョン管理機構はそのため基本メカニズムである。しかしながら、ソフトウェアが進化出来るか否かは、それが如何に素直に実世界を抽象化しているかにかかっている。この意味で、ソフトウェアオブジェクトベースは強力な抽象化メカニズムとその進化のメカニズムを持っている必要があるが、これについては我々は多くを知らない。

5. おわりに

ソフトウェアオブジェクトベースに求められる要件のうち特に重要なものについて考えてきた。もちろんこれだけが重要というわけではなく、上記のもの以外にも、例えば、設計データベースにつきものの長時間トランザクション、オブジェクトの型付けと型進化、オブジェクトの複数表現、分散オブジェクト管理、アクセスコントロールなど解決すべき問題が多数指摘されている。これらの問題の解決はソフトウェア開発環境にとって重要なだけでなく、新しいデータベース研究にも有用な示唆を与えるもの考えられる。

抽象化と局所性

慶應義塾大学理工学部

所 真理雄

開放型分散環境上での計算においては、抽象化と同時に局所性を考えることが重要である。そこでは抽象化はプログラミング時の宣言や指定で静的に行なわれるものではなく、実行時に動的に行なわれる。局所性にはいくつかの視点がある。1つは、あるオブジェクトが一時に関連するオブジェクトの数は有限であるという点である。他は、分散システムにはユニークなグローバルビューは存在せず、情報を得るには行動（計算）を伴い、従って時間がかかるという点である。すなわち、抽象化は実行を伴い、実行は局所性に制限され、局所性は抽象化による。

次世代データベースシステム宣言に向けて —ソフトウェア・オブジェクトベース構築の立場から—
オブジェクトベースはソフトウェア開発過程を強く支援しうるか？

徳田雄洋 (東京工業大学工学部情報工学科)

1. はじめに

1.1 オブジェクトベースの出現

”データベースとオブジェクト指向プログラミング言語の連携”へと、データベースとオブジェクト指向プログラミング言語のそれぞれが発展した結果、オブジェクト指向データベース(以下、オブジェクトベースと呼ぶ)が登場した。

オブジェクトベースが、オブジェクト指向プログラミング言語をホスト言語とするデータベースアプリケーションに有用なことは自然であろう。しかしながら、ソフトウェア開発過程、特に一般の言語を用いるソフトウェア開発過程を強く支援するデータベース構築のための強力なデータモデルになりうるかどうかはまだ明らかではない。オブジェクトベースは、ソースコード管理と実行形式製作手順記述に、基本的に留まっているソフトウェアデータベースを、質的に大きく変化させうるのだろうか？

1.2 オブジェクトベースの定義

残念なことにオブジェクトベースという用語は目下の所きわめてあいまいである。何をもちてオブジェクトベースとするかは、用語を使用する人の強調部分によって次のように大きく異なってくる。

- 1) 従来のデータベースのデータ記述とデータ操作に、オブジェクト指向プログラミング言語の記法を使うデータベース。
- 2) オブジェクト指向のメッセージ型計算システムにおいて、セッションを越えて存在する永続的オブジェクトや1セッション全体の状態を保存する記憶場所。
- 3) いつでもメッセージを受けると動作し、ずっと存在し続けるメッセージ型計算システムそのもの。
- 4) ユーザやシステムが定義した抽象データ型のデータオブジェクトを保存し取り出す記憶場所。
- 5) 抽象的には、それぞれ属性を持っている点と点同士を有向矢印で結んでできる有向グラフのこと。

6) プログラム、テキスト、音声、図といったマルチメディアのオブジェクトが自由に貯蔵できて、しかも容易に取り出せる記憶場所。

7) 値指向でないデータベース一般のこと。

2. ソフトウェアデータベースに何が必要か？

ソフトウェアデータベースに必要な機能をまとめると以下ようになる。

- 1) ソフトウェア作成過程で生じるさまざまな生成物を保存し、取出せること。
- 2) ソフトウェア作成過程で生じるさまざまな生成物に対し、照会を行うと回答が得られること。
- 3) 生成過程の記述を保存し、実行できること。
- 4) 同一生成物のたくさんのバージョンを管理できること。これらから意味のある組合せを取り出せること。
- 5) ソフトウェアプロジェクトの所属者および外部ユーザの権限を明確に管理できること。
- 6) ソフトウェアプロジェクトの所属者複数による並行トランザクションを管理できること。
- 7) 障害から最近の正しい状態まで回復できること。

3. オブジェクトベースの例題

データベース問題の典型的例題としては、例えば以下の4つがある。

- 1) 従業員給与管理
- 3) 学生進級管理
- 2) 図書館書籍管理
- 4) ソースコード管理

オブジェクトベースとして構成する場合どうい問題があるかを、上記の4例について簡単にみてみよう。

1) 従業員給与管理

例えば、従業員1人1人を1オブジェクトとすると、特定の氏名の人が何人いるかを、従業員集合オブジェクトに問い合わせ、ポインタの値をたどって、各オブジェクトの氏名に到達することになると思われる。

2) 図書館書籍管理

1つの本を1オブジェクトとすると、本文部分もし

まうことが考えられる。この場合、本のテキストや図の部分の照会、通常のデータベース的な照会でなくなる。そこで、例えばテキストオブジェクトが複雑なテキスト処理のメソッドを持つように作ることに思われる。

3) 学生進級管理

学生進級管理を難しくしている理由に次のようなものがある。

学年毎に異なる進級条件規則の存在、学年毎に異なるうる学科組織、学年毎に異なるカリキュラム、これらの異なった学年を通過する一部留年生の存在。

方針としては、学生の1人1人にそれぞれ有効な進級条件判定のメソッドを持たせる。こうすると外部の照会は一様な方法でアクセス可能となる。

しかし、学科が2コースに分割した新学年に、留年生が降りてきた場合の所属の扱い等は依然問題である。

4) ソースコード管理

ソースコードのどの単位を1オブジェクトとするのかを決め、ソースコードを蓄える。ソースコードの保存そのものには余り困難はないが、ソースコードについての照会の回答は簡単ではない。小さなオブジェクトから大きなオブジェクトまで照会回答のためのメソッドを提供する必要がある。またいかなる質問形式を許し、それぞれの質問にどのように前処理結果を利用、あるいは前処理結果の利用なしに直接的に、答えるかが問題である。

4. ソフトウェアデータベースとしてのオブジェクトベース

以下のようにいくつかの検討課題がある。

- 1) 機械の中の現実世界としてソースコードは存在するが、これからデータベース表現をどう取り出すのか。また一般にソースコードの変更は頻煩に起こりうる。
- 2) 如何なる基本単位を1オブジェクトとするのか。大きなオブジェクトから小さなオブジェクトまで記憶管理を行う必要もある。
- 3) 照会のためにどのようなメソッドを各オブジェクトに持たせるのか?
- 4) 深い階層関係を表現する必要がある。部分と全体の関係は、参照可能な外部的存在物へのポインタとして表現されることになる。

て表現されることになる。

- 5) 1つのトランザクション時間が従来のデータベースのトランザクション時間より普通ずっと長い。
- 6) 提供されるメソッドの集合と照会文の構文規則から、回答可能な照会文の形式を特徴付ける必要がある。
- 7) 人間のソフトウェア作業の失敗からの回復では、いつを失敗とし、どこまで回復させるかは一般には容易に決まらない。

5. 結論

- 1) ソフトウェアデータベース構築の諸問題の中で、オブジェクトベースというデータモデルを採用することによって単純化されうる問題は部分的範囲にとどまると思われる。
- 2) オブジェクト指向の設計方法論では、一応どういう単位をオブジェクトの単位として扱っても自由ということになる。これはソフトウェアデータベース設計者にとって逆に扱いにくい場合がある。
- 3) 電子辞書、知識ベース、テキストベースといった分野では必ずしもデータベースの一般論とは異なる固有のアプローチが取られている。ソフトウェアデータベースでもそういった発展の可能性がある。

参考文献

- [1] 石原博史・徳田雄洋: 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、情報処理学会研究報告 89-PL-21 (1989)
- [2] 石原博史・徳田雄洋: 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、電子情報通信学会研究報告 SS-89-27 (1990)
- [3] ソフトウェア工学研究財団: ソフトウェア開発のためのデータベース、ソフトウェア工学における国際共同研究・交流のあり方に関する調査研究報告書 (1990)
- [4] 富永和人・徳田雄洋: 新しいソフトウェアデータベースを目指して、本情報処理学会研究報告 (1990)
- [5] 吉沢克典・徳田雄洋: C言語によるソフトウェア開発の管理システムCORONA、日本ソフトウェア科学会第5回大会論文集 41-44 (1988)

Events Models for Integration

David Notkin

Department of Computer Science & Engineering, FR-35
University of Washington
Seattle, WA 98195 USA¹

One important role for modern software object bases is to simplify the construction and evolution of integrated software environments. One part of a software database is the event model, which describes how objects pass control indirectly. In an event model, objects announce events that indirectly invoke specified methods in zero or more interested objects. Examples include the Smalltalk-80 change/update protocol (upon which the Model-View-Controller is built [Krasner & Pope 88]), and AP5, which invokes specified functions when arbitrary relational predicates are satisfied [Balzer 85].

An event model with specific properties makes it much easier to define, understand, and change interactions among the objects in an integrated environment. This position paper describes (1) a set of limitations that an event model must overcome to properly support integration, and (2) a set of alternatives for managing the relationship among events and objects. These ideas form the basis of an event model that eases the construction and evolution of integrated software environments [Sullivan & Notkin 90].

Limitations to Overcome. The first limitation is when events are not explicitly declared as part of the specification of objects. FIELD [Reiss 88] and the Smalltalk-80 change/update protocol have this limitation. Since events are critical aspects of object behavior, they must be included in the object's specification.

The second limitation restricts which objects can announce events. Only relations announce events in APPL/A [Heimbigner, Sutton & Osterweil 87], for example, while in Smalltalk-80, in contrast, all objects can announce events. Limiting the objects that can announce events limits the kinds of objects that can be integrated.

The third limitation restricts which events objects can announce. For example, Gandalf's [Habermann & Notkin 86] action routines are invoked based on about a dozen kernel-defined events; similarly, active values in LOOPS [Stefik, Bobrow & Kahn 86] announce events only when they are read or written. FIELD tools, in contrast, announce arbitrary events by sending ASCII messages. At best, this limitation makes programming with events difficult and hard to understand; at worst, it constrains the kinds of relationships that can be based on events.

Managing Event Associations. Any event model that overcomes these limitations must still answer at least one major question: How are the associations between events and objects created, removed, and managed? Abstractly, these associations can be characterized in terms of a relation *EM*. If an event-method tuple (*e*, *m*) is in *EM*, when *e* is announced *m* is invoked. The order and synchronization of invoking multiple methods associated with a given event must be specified by each event model.

The interface to *EM* must provide at least register, unregister, and lookup operations. The register and unregister operations add or remove tuples from *EM*. The lookup operation finds all *m*'s to call when a specific *e* is announced. *EM*'s interface may also

¹Currently on leave at the Department of Information and Computer Sciences, Osaka University, Machikaneyama 1-1, Toyonaka City, Osaka 560 JAPAN.

support richer operations. For example, attribute grammar preprocessors use an interface to *EM* that provides query operations that check for circularities. Rich operations like these could support integration by enabling runtime queries such as: "What methods will be invoked if I announce the 'breakpoint at line 12' event?"

Explicitly inserting event-method tuples into *EM* is one way to handle registration. In attribute grammar systems, for instance, this relationship is usually computed during preprocessing. Registration can also be handled by computing (part of) *EM* when an event is announced. In FIELD, for instance, *EM* is represented as a set of regular-expression/method pairs; when an (ASCII) event is announced, all regular-expressions that match that event invoke their associated method.

A central agent can be used to manage *EM*, or individual objects can manage associations between events and methods. This decentralized style generally makes the basic register, unregister, and lookup operations fast, since *EM* is (in essence) partitioned into subrelations. The centralized style, in contrast, generally makes the basic operations more expensive, but it also eases the definition of richer operations on *EM*.

No single set of decisions about these (and other related) issues is demonstrably superior to others. Different domains and style of environments will demand different solutions.

Conclusions. Thus, there are two lessons in this position paper about defining software object bases that are to support integration. The first is that its event model must overcome the three limitations listed above. The second is that in order to support domains with different requirements, the event model must allow for diverse approaches to event-method management.

References

- [Balzer 85] R. Balzer. A 15 Year Perspective on Automatic Programming. *IEEE Trans. Softw. Eng. SE-11*,11 (Nov. 1985).
- [Habermann & Notkin 86] A.N. Habermann and D. Notkin. Gandalf Software Development Environments. *IEEE Trans. Softw. Eng. SE-12*,12 (Dec. 1986).
- [Heimbigner, Sutton & Osterweil 87] D. Heimbigner, S. Sutton, and L. Osterweil. APPL/A: A Language for Managing Relations Among Software Objects and Processes. Univ. Colorado Tech. Report CU-CS-374-87 (1987).
- [Krasner & Pope 88] G.E. Krasner and S.T. Pope. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *J. Obj. Oriented Progr.* 1,3 (Aug./Sep. 1988).
- [Reiss 88] S. Reiss. Integration Mechanisms in the FIELD Environment. Brown Univ. Tech. Report CS-88-18 (1988).
- [Stefik, Bobrow & Kahn 86] M.J. Stefik, D.G. Bobrow, and K.M. Kahn. Integrating Access-Oriented Programming into a Multiparadigm Environment. *IEEE Software* (Jan. 1986).
- [Sullivan & Notkin 90] K. Sullivan and D. Notkin. Reconciling Integration and Independence. To appear, *Proc. ACM SIGSOFT90: 4th Symp. Softw. Development Env.* (Dec. 1990).