リレーションを陽に扱う、プロトタイプによるオブジェクト指向言語 *Eunice97*

河田 恭郎 Andrew E. Santosa 前川 守
電気通信大学大学院情報システム学研究科

我々が進めてきていたEuniceプロジェクトの*Eunice97*言語を概観する。*Eunice97*はプロトタイプによるオブジェクト指向言語であり、リレーションをファーストクラス・オブジェクトとして扱えるのが最大の特徴である。通常のオブジェクト指向言語では組み込み機能となっている類形化／個別化リレーションもリレーションの一つであり、それによってメタプログラミング機能が提供される。

リレーションは階層的に定義できる。リレーションを利用することで、概念モデリングが容易になり、また、用途に応じたリレーションを用意することで、記述がより正確になる、などの利点が得られ、ソフトウェア開発の上流工程からの支援をめざすEuniceプロジェクトの目的も合致する。

Eunice97: A Prototype-Based OO Language with Relations as First-Class Objects

Yasuro Kawata, Andrew E. Santosa, and Mamoru Maekawa
Graduate School of Information Systems, University of Electro-Commnications

This paper presents an overview of the *Eunice97* language, which is to play the key role in our Eunice Project. *Eunice97* is a prototype-based object-oriented language, and has relations as first-class objects. This applies even to the *IS-META-OF* relation, which is usually embodied as instantiation mechanism in conventional object-oriented languages. Meta programming facilities is provided along this *IS-META-OF* relationship.

Relations are hierarchically structured. Relations facilitate conceptual modeling, and caters to more accurate description because each and every relation required can be defined. These features are especially vital to the Eunice Project, which places emphasis on support for upperstream software development.

1 はじめに

本稿では、我々が進めてきていたEuniceプロジェクト[4]を簡単に紹介した後、その第IIフェーズの核となる、*Eunice97*言語の特徴について概観する。

2 Eniceプロジェクト

ソフトウェア工学の中心的課題とは、大規模ソフトウェア開発をいかに容易にするか、ということであろう。我々が進めているEuniceプロジェクトではこの課題に、仕様記述から設計、実装までを支援する単一言語を提供する、というアプローチで取り組む[4]。

開発段階を明確に分け、またそれが分けられるという前提に依存しているウォーターフォール開発モデルは、古典的といわれつつも、現実にはそれに替わるような決定的な新モデルや手法の出現をいまだ許していない。ウォーターフォール・モデルによる開発では：

- 開発段階ごとに記述体系が異なるために、段階間で意味情報が欠落してしまう。
- 後段階になって前段階になんらかの問題が発見されたときにも、前段階を修正せぬまま後段階を進めてしまうことが多いため段階間で整合性がなくなってしまい、結局ソースコードしか信用できない。
- そもそも仕様と実装を厳密に区別することができない。必然的に両者は"絡み合う"ことになる[17]。

といった問題が原理的に常に存在する。これを根本的に解決するには、上記のような単一言語（図式言語も含む）を用意するしかない。そのような言語として、我々は*Eunice*言語を設計してきた[1]。

*Eunice*言語がそのような言語である以上、その特徴の一つは、任意の開発段階で実行可能であることになる。すなわち、*Eunice*は実装言語でもあり、*Eunice*で記述された仕様もすべて実行可能である。このことは*Eunice*が、Z [22, 23]に代表される形式的仕様記述言語に属さず、PAISLey [24, 25]に代表される実行可能仕様記述言語の流派に属することを意味する。

そのため、通常の実装用プログラミング言語と比較して、*Eunice*言語に対して特に以下のような要請がある[2]：

A. 開発の上流工程の支援

プログラミング言語は、ソフトウェア開発工程のうち実装段階を支援する。要求仕様抽出や仕様記述の段階などの開発の上流工程は通常、各種開発手法やそれに準拠したCASEツールにより直接的・間接的に支援されている。

しかし、*Eunice*はこれらの工程も支援しなくてはならない。

B. 高い抽象度と豊富なプリミティブ

A.とも関係があるが、上流工程での記述は、詳細が決まっていないため、自然と抽象度が高くなる。そのような抽象度の高い記述は、表現したい内容が過不足なく簡潔に表現できる、すなわち、豊富なプリミティブがあらかじめ用意されているときに初めて可能になる。

C. 柔軟性

これもA.と関係がある。特に上流工程においては、あいまいであったり、矛盾があったりする記述になる。それが検出できることはもちろん重要であるが、それと同時にそのような状態でも可能な範囲で実行でき、また、ユーザが望むような変更が任意の時点でできることが必要である。

D. 拡張性

ソフトウェアのライフサイクルの中では、通常先に述べたように多くの言語（図式言語も含めて）が情報の表現媒体として利用される。我々は*Eunice*がこれら全てに代わることを目標としているのだが、単純に

[1] プロジェクト名と言語名が同じなので紛らわしいが、本稿では前者をローマン体にし、後者をスラント体にすることで区別する。

[2] これらはそれぞれ独立の要請ではなく、互いに密接に関係し合っていることに注意。

あらゆるものを取り込もうとすると言語仕様が膨大になってしまう。

そのような言語は、各適用領域、あるいは、同じ適用領域でも開発グループの今までの慣習などによって、ほぼ無限のバリエーションがある。それを考慮しても、あらかじめ*Eunice*に取り込もうとするのは、やはりはなはだ非現実的なアプローチである。

したがって、*Eunice*の基本言語仕様は極力小さく保つ一方、強力な拡張機能を持たせ、さまざまな要求に答えられるようにするようにするのが唯一現実的なアプローチである。

一見B.の逆のようであるが、どちらも必要である。

3 *Eunice97*

本説では、我々が現在設計を進めている*Eunice*言語の最新バージョン*Eunice97*の暫定的な仕様について述べる[3]。

3.1 特徴

*Eunice97*は、前節で述べたような要請を満たすべく、以下のような特徴を持つ。

1. プロトタイプによる (prototype-based) オブジェクト指向

   世の中で広く使われているオブジェクト指向言語、は、すべてクラスによる (class-based) オブジェクト指向を採用している。しかし、*Eunice97*はクラスによるオブジェクト指向ではなく、プロトタイプによるオブジェクト指向を採用する。すなわち、オブジェクトとはクラスなるテンプレートから生成されるのではなく、既存のオブジェクトをクローンすることで生成される。

   また、プログラム中に存在するエンティティは、全てオブジェクトとして表現されうる。例えば、通常のオブジェクト指向言語ではオブジェクトの一部とみなされるメソッドなども全てオブジェクトとして扱うことが可能である。これは2.からの要請でもある。

2. メタオブジェクト・プロトコル

   メタオブジェクト・プロトコル (MetaObject Protocol, MOP) とは、オブジェクト指向の文脈における制御されたリフレクション（自己反映）機構である。*Eunice97*は、言語自身の各種拡張と、開発環境の各種ツールとの連携のためにMOPを利用する。

3. リレーションをファーストクラス・オブジェクト (first-class objects) して導入

   他のオブジェクト指向言語・システムでは、MOPを支援するシステムでさえ汎化 (generalization) / 特化 (specialization) リレーション、あるいは類形化 (classification) / 個別化 (instantiation) リレーションは、最初からその存在を仮定している。具体的には、前者は通常クラス間のインヘリタンス機構として実現し、後者は同じく通常クラスとそれから生成されるインスタンスとして実現している。*Eunice97*においては、これら基本的なリレーションも汎用のリレーションの一種に過ぎない。

以上述べたような特徴により、*Eunice97*は以下のような利点を持つことになる：

- 開発初期の支援が十分に行なえる（A.）。オブジェクト・モデリング・テクニーク (OMT)[14] に代表されるオブジェクト指向開発手法では、最初にオブジェクトの抽出を行なうのが通常である。この段階では「名詞で表現されるのがオブジェクト、動詞で表現されるのがメソッド」などという大雑把な指標しかない。また、何がクラスで何がインスタンスなのかの区別も決して自明ではない。微妙なノウハウは存在しても、オブジェクトの抽出からそれによるオブジェクト・モデルの構築までを、最初か

[3] *Eunice97*は、我々が過去において発表した古い版の*Eunice*（例えば[5]）とは、設計思想から根本的に異なること、また、本稿の*Eunice97*の仕様もあくまで暫定的なもので、近い将来大きな変更の可能性もあること、を御留意頂きたい。

ら完全に行なえることを期待するのは現実的ではない。

しかし、*Eunice97*の世界では、1.から、この種の「誤り」はもはや本質的な誤りではない。クラスもインスタンスも、スロットもメソッドもリレーションも、最初は全て、「よくわからないがともかくオブジェクト」という段階の認識も表現できる。また、一旦クラス（相当）と認識したオブジェクトを、インスタンス（相当）と認識し直したりするようなことまで可能である（C.）。

ソフトウェア開発の上流工程を支援するCASEツール（例えば前述のOMTに準じたもの）においても、これほどまでの柔軟性は持っていない。このようなツールでは、例えばクラスとインスタンスの区別は絶対であり、容易に変更できない。

クラスベースのオブジェクト指向言語・システムと比較して、プロトタイプ・ベースのそれの利点はいろいろあるが、ソフトウェア開発の上流工程、特に解析の段階までの利用をうたったものはない。

- 前項とも関係あるが、概念モデリングが行ないやすい（A.およびB.）。オブジェクトとその間に成立するリレーションによる概念/世界モデリングは、データベースの分野では、エンティティ-リレーションシップ・モデル（ERモデル[2]）として現れ、知識処理の分野では意味ネットワーク[10]（後のフレーム表現[6]のもと）として現れたアイディアの自然な延長である。現に多くのオブジェクト指向開発手法で採用されており、その有効性は広く認められている。

  *Eunice97*は、通常のオブジェクト指向言語において、クラス間の継承、クラスをテンプレートとしたインスタンス生成、オブジェクトのスロット、などの機構を通じて陰に提供されている標準的なリレーションはもとより、任意のリレーションを支援する(3.)ことでこれを可能にする。リレーション間のリレーション、すなわち、高階リレーションをも表現できる。高階リレーションにより、推移性を持つリレーション、反射的なリレーションまで表現でき、その表現力は極めて豊かである。

- これも前項と関係ある。インヘリタンスは、オブジェクト指向言語・システムの特徴の一つと通常数えられるが、実はその定義は多種多様である。しかも、インヘリタンスは、概念の特殊化、コードの再利用、インタフェースの共有など、通常複数の意味的関連を負わされている（[18]がよくまとめている。）

  どのようなインヘリタンスを用意するかが、オブジェクト指向言語・システムの設計において常に問題となっていたが、*Eunice97*においてはそれは問題にならない。なぜなら、表現したい意味的関連に1対1に対応するリレーションを定義すればよいからである。この機能により、*Eunice97*による記述は、非常に概念的に明確になる（B.）。

- プロトタイプによるオブジェクト指向言語では、クラスによるオブジェクト指向言語に比較して、柔軟である代わりに記述の抽象度が低くなってしまうことがある。

  例えば、Self [20]は、非常に単純なオブジェクト・モデルを持つ：オブジェクトは、通常のオブジェクト指向言語におけるインスタンス変数とメソッドを統合したスロットと、他のオブジェクトを指すペアレント・リンクからなる。ペアレント・リンクを手繰って委譲（delegation）が行なわれ、これにより振舞いおよびデータの共有が実現できる。

  このモデルは単純でわかりよく、[19]にあるように、クラス・ベースのオブジェクト指向言語では実現が困難なことも、容易に実現できる。しかし、それには代償がある。

  クラスによるオブジェクト指向言語であるなら単一オブジェクトのクラスとして表現されることが、Selfではtraitsオブジェクトとprototypeオブジェクトの二つで表現される。一般に、プロトタイプによるオブジェクト指向言語の記述の抽象度の低さから、ペアレント・リンクでつながれた複数オブジェクトの集まりとして表現し

なければいけないことが多い。これを「分離したオブジェクト (split objects)」という [3]。これでは、自分がコードを書くときは楽だが、他人のコードを読むのは困難である。

これらをどう扱うかが、プロトタイプによるオブジェクト指向言語における一つの最近の研究課題となっている（例えば [1] や [11]）が、*Eunice97*においてはそれはどのようなリレーション・オブジェクトを用意するかという問題に帰着する。すなわち、**3.**が**1.**の欠点を補っているわけである。

- オブジェクトに対する、copy のような基本的操作は、一般に shallow copy と deep copy を必要に応じて利用する。しかし、どちらがふさわしいかは、リレーションの性質として定義できるものである [13]。オブジェクトに対するそのような操作がどのように他のオブジェクトに影響するかを、リレーションの性質として記述できる。

3.2 *IS-META-OF*リレーションとメタ・リグレッション

前述したように、*Eunice97*は類形化 / 個別化リレーション（以降簡単のため、「*IS-META-OF*リレーション」）や、汎化 / 特化リレーション（同様に「*IS-SUPER-OF*リレーション」）もリレーションの一種類として扱う。

図 1に示すように、*IS-META-OF*リレーションに関して、容易にメタ・リグレッションが起きるがこれはこの種のシステムの本質的な制約なのでやむを得ない。

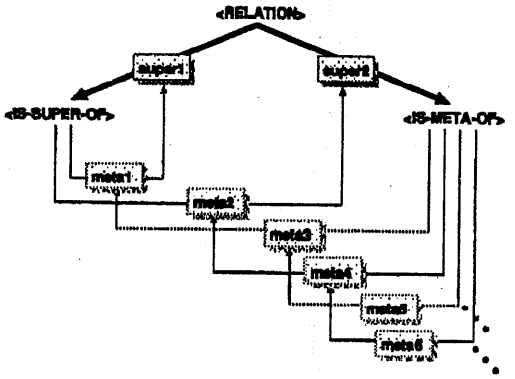

図 1: *Eunice97*メタ・リグレッション

関連研究

明示的にリレーションを採り入れたオブジェクト指向言語としては、Rumbaugh らによる DSM [15, 13, 12] がある。ただし、リレーションはファーストクラス・オブジェクトではないし、メタ・プログラミングが可能なわけでもない。しかしその他の点については *Eunice97*は DSM より大きく影響を受けている。*Eunice97*は [12] にあげられている大半の問題を解決したことになる。

まとめ

本稿では、我々が進めてきていた Eunice プロジェクトの目的を述べた後、それを実現する上での核となる、*Eunice97*言語の特徴について概観した。

その大きな特徴は、特に上流工程の支援を見込んで、柔軟かつダイレクト・マニピュレーションに優れたプロトタイプ・ベースのオブジェクト指向を採用しながらも、リレーションをファーストクラス・オブジェクトとすることで、精密な記述を可能にしていることである。その期待される利点は多岐に渡る。

参考文献

[1] Daniel Bardou and Christophe Dony. Split objects: a disciplined use of delegation within objects. In OOPSLA96 [9], pp. 122–137.

[2] Peter Pin-Chan Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, Vol. 1, No. 1, pp. 9–36, 1976. Reprinted in [16].

[3] Christophe Dony, Jacques Malenfant, and Pierre Cointe. Prototype-based languages: From a new taxonomy to constructive proposals and their validation. In OOPSLA92 [8], pp. 201–217.

[4] Y. Kawata, A. Yabu, M. Maekawa, H. Kobayashi, T. Kawase, and H. Kozuka. A perspective on a Eunice-based control system development environment. In Qiangnan Sun, Zesheng Tang,

and Yijun Zhang, editors, *Computer Applications in Production Engineering: Proceedings of CAPE '95*, pp. 540–547. International Federation for Information Processing (IFIP), Chapman & Hall, 1995.

[5] Yasuro Kawata, Hisahiro Kobayashi, Akifumi Yabu, Kimiya Onogawa, Akira Kawasaki, and Mamoru Maekawa. Eunice/ITRON: A control system development environment for ITRON machines. In *Proceedings of the 11th TRON Project International Symposium*, pp. 91–105. IEEE Computer Society Press, 1994. December 7–10, 1994, Tokyo, Japan.

[6] Marvin Minsky. A framework for representing knowledge. In Winston [21], pp. 211–277.

[7] *ACM SIGPLAN Notices*, Vol. 22, No. 12, December 1987. Norman Meyrowitz, *ed.*, Proceedings of OOPSLA '87, October 4–8, 1987, Orland, Florida, USA.

[8] *ACM SIGPLAN Notices*, Vol. 27, No. 10, October 1992. Andreas Paepcke, *ed.*, Proceedings of OOPSLA '92, October 18–22, 1992, Vancouver, British Columbia, CANADA.

[9] *ACM SIGPLAN Notices*, Vol. 31, No. 10, October 1996. Proceedings of the 1996 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA '96). San Jose, California, USA, October 6–10, 1996.

[10] M. R. Quillian. Semantic memory. In *Semantic Information Processing*, pp. 227–268. MIT Press, 1968.

[11] Hernán Astudillo R. Reorganizing split objects. In OOPSLA96 [9], pp. 138–149.

[12] James Rumbaugh. Relations as semantic constructs in an object-oriented language. In OOPSLA87 [7], pp. 466–481. Norman Meyrowitz, *ed.*, Proceedings of OOPSLA '87, October 4–8, 1987, Orland, Florida, USA.

[13] James Rumbaugh. Controlling propagation of operations using attributes on relations. *ACM SIGPLAN Notices*, Vol. 23, No. 11, pp. 285–296, November 1988. Norman Meyrowitz, *ed.*, Proceedings of OOPSLA '88, September 25–30, 1988, San Diego, California, USA.

[14] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling And Design*. Prentice Hall, 1991. (日本語訳) 羽生田栄一監訳, 「オブジェクト指向方法論OMT—モデル化と設計」, トッパン, 1992.

[15] Ashwin V. Shah, James E. Rumbaugh, Jung H. Hamel, and Renee A. Borsari. DSM: An object-relationship modeling language. *ACM SIGPLAN Notices*, Vol. 24, No. 10, pp. 191–202, October 1989. Norman Meyrowitz, *ed.*, Proceedings of OOPSLA '89, October 1–6, 1989, New Orleans, Louisiana, USA.

[16] Michael Stonebraker, editor. *Readings in Database Systems*. Series in Data Management Systems. Morgan Kaufmann Publishers, second edition, 1994.

[17] William Swartout and Robert Balzer. On the inevitable intertwining of specification and implementation. *Communications of the ACM*, Vol. 25, No. 7, pp. 438–440, July 1982.

[18] Antero Taivalsaari. On the notion of inheritance. *ACM Computing Surveys*, Vol. 28, No. 3, pp. 438–479, September 1996.

[19] David Ungar, Craig Chambers, Bay-Wei Chang, and Urs Hölzle. Organizing programs without classes. *Lisp and Symbolic Computation*, Vol. 4, No. 3, pp. 37–56, June 1991. Published from Kluwer Academic Publishers.

[20] David Ungar and Randall B. Smith. Self: The power of simplicity. In OOPSLA87 [7], pp. 227–242. Norman Meyrowitz, *ed.*, Proceedings of OOPSLA '87, October 4–8, 1987, Orland, Florida, USA.

[21] P. Winston, editor. *The Psychology of Computer Vision*. McGraw-Hill, 1975.

[22] J. C. P. Woodcock, editor. *Using Standard Z: Specification, Proof and Refinement*. International Series in Computer Science. Prentice Hall, Hemel Hempstead, Hertfordshire, UK, 1993.

[23] J. B. Wordsworth. *Software Development with Z: A Practical Approach to Formal Methods in Software Engineering*. International Computer Science Series. Addison-Wesley, 1992.

[24] Pamela Zave. An operational approach to requirements specification for embedded systems. *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 3, pp. 250–269, May 1982.

[25] Pamela Zave. An insider's evaluation of PAISLey. *IEEE Transactions on Software Engineering*, Vol. 17, No. 3, pp. 212–225, March 1991. Special section on requirements engineering.