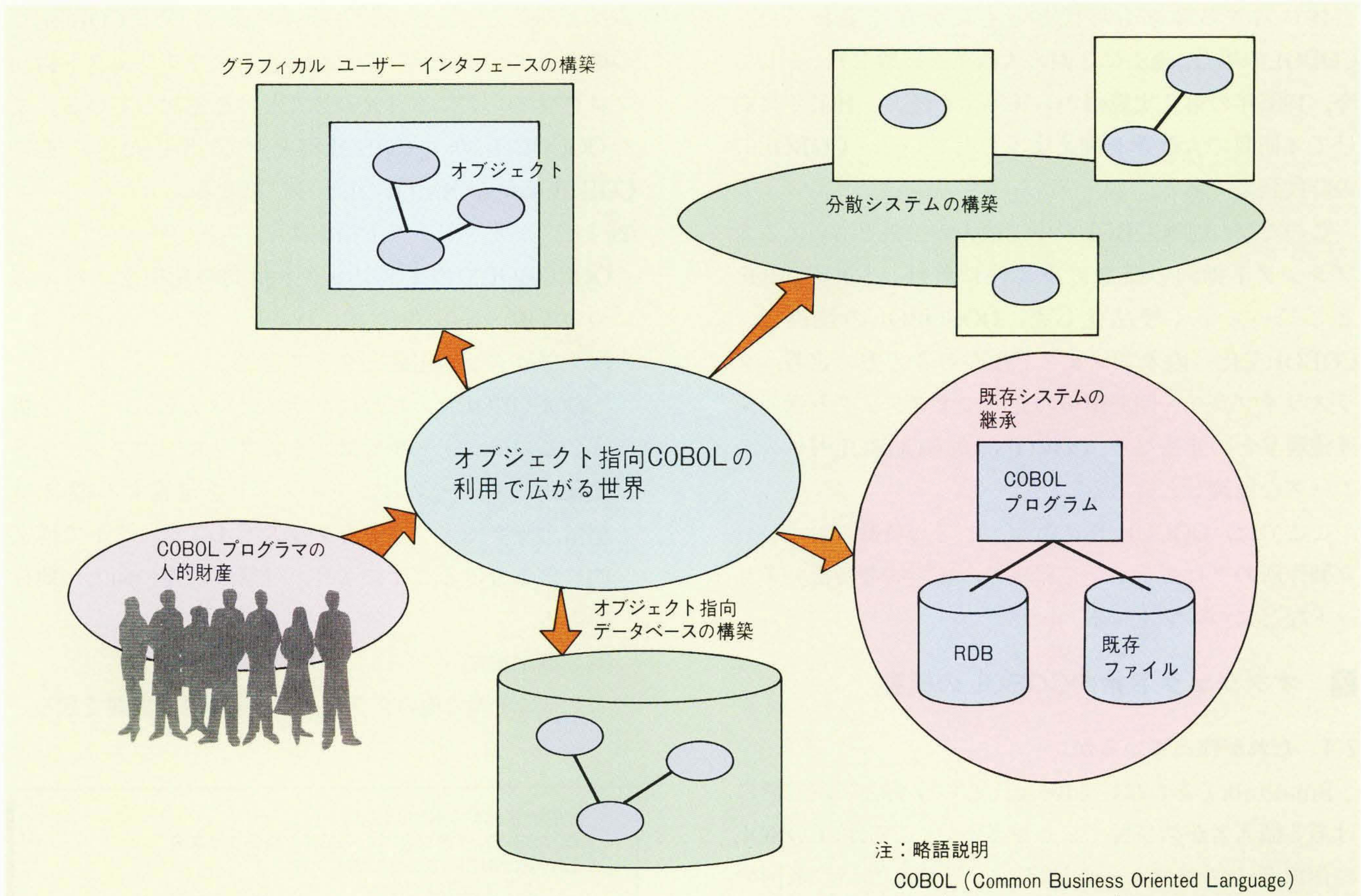
プログラミング環境を充実させたオブジェクト指向COBOL

Programming Environments for Object-Oriented COBOL

西尾高典*　*Takanori Nishio*　横塚大典*　*Daisuke Yokotsuka*
前田秀子*　*Hideko Maeda*　権藤文代*　*Fumiyo Gondô*



オブジェクト指向の世界へのゲートウェイ「オブジェクト指向COBOL」
オブジェクト指向COBOLにより，過去のプログラム・データ資産と最新のオブジェクト指向技術との融合を図ることができる。また，COBOL85と完全上位互換であるため，徐々にオブジェクト指向を取り込むことができる。

このたび，次期標準COBOL97の「プリビュー」ともいうべきオブジェクト指向COBOL(以下，OOCOBOLと言う。)の本格的なプログラミング環境を，Windows NT[※1]およびUNIX[※2]で製品化した。

この環境は，開発マネージャやブラウザ，COBOLエディタ，アニメータ機能を持つデバッガ，ファイル・レコード定義，スクリーン定義機能など，COBOLプログラム開発向けの充実した機能を提供している。COBOL85と完全上位互換であるため，徐々にオブジェクト指向を取り込めると同時に既存資産もそのまま活用できる。オブジェクト指向データベースや分散オブジェクト環境などの新しいパラダイムの考え方とも親和性が良い。また，RDBやOLTP(Online Transaction Processing)などの各種ミドルソフトウェアや上流CASE(Computer Aided Software Engineering)，GUI(Graphical User Interface)構築ツールとも連携しており，企業情報システム開発の中核としての活躍が期待できる。

* 日立製作所 ソフトウェア開発本部

1 はじめに

COBOLが汎(はん)用計算機用プログラミング言語として誕生してから30年以上がたち，その間に計算機も，これを利用する社会も大きく変貌(ぼう)を遂げた。言語自体に対する要求も時代とともに大きく変わった。COBOLの場合，過去に3回の大改革を経験した。そして今，1985年の第3次規格から10年余を経て，1997年に向けて4回目の大改革を迎えようとしている。COBOL97の時代到来である。

このたび，次期COBOLの中で最大の拡張ともいえるオブジェクト指向の概念の重要性に着目し，OOCOBOLとしていち早く製品化した。OOCOBOLの役割は，COBOL文化へのオブジェクト指向の考え方の定着，クラスライブラリ・開発環境の充実，オブジェクト指向関連他製品との連携など，COBOL85からCOBOL97へのスムーズな橋渡しと言える。

ここでは，OOCOBOLの言語としての特徴，および日立製作所のプログラミング環境での機能的な特徴，メリットなどについて述べる。

2 オブジェクト指向COBOLの概要

2.1 だれが作っているか

SmalltalkであればGoldberg氏という具合に，言語にはある個人名が結び付くことが多い。ところが，COBOLの作成者のプロフィールはほとんど見えない。米国のCODASYL(Conference on Data System Languages)が制定してきたからである。現在は米国の標準化団体ANSI(American National Standards Institute)傘下の委員会X3J4が仕様の検討などを行っている。ドラフトがほぼ固まりつつあり，舞台はISO(International Standardization Organization)へと移りつつある。

X3J4にはわが国の情報処理学会規格調査会も加入している。米国の委員会でありながら，わが国もその仕様確定に寄与している。この調査会のSC(Subcommittee) 22 COBOLWG(Working Group)の主査は日立製作所が務める。米国の委員会にも積極的に参加しており，COBOLの標準化に貢献している。

2.2 言語の特徴

(1) COBOL85との上位互換性

OOCOBOLの最も大きな特徴は，従来のCOBOLとの完全上位互換性であろう。800億ステップと言われる過去の膨大なプログラム資産を生かしつつ，新しいパラダイムへと進化できる。上位互換性を保つためにCOBOL85の構文をすべて継承しており，さらにオブジェクト指向プログラミングに不可欠の2文だけを追加している。またOOCOBOLからCOBOL85を呼び出したり，逆にCOBOL85からOOCOBOLを呼び出せる。

(2) どこがオブジェクト指向か

OOCOBOLでのオブジェクト指向の諸概念の扱い方について述べる。

(a) プログラム定義とクラス定義

OOCOBOLは，新たにクラスというモジュールを扱う。このモジュールでは，そのクラスやオブジェクトが使用するデータおよびメソッドを定義する(**図1参照**)。データやメソッドをクラスとオブジェクトで構文的に書き分けることにより，理解容易性の向上が期待できる。

(b) 継承機能

クラス定義で他のクラスとの継承関係が定義できる。

```
IDENTIFICATION DIVISION.
CLASS-ID. クラス名　INHERITS 祖先クラス名.
ENVIRONMENTS DIVISION.
    クラスの環境部本体

    IDENTIFICATION DIVISION.
    CLASS-OBJECT.
    ENVIRONMENTS DIVISION.
        クラスオブジェクトの環境部本体
    DATA DIVISION.
        クラスデータ定義
    PROCEDURE DIVISION.
        クラスメソッド定義
    END CLASS-OBJECT.

    IDENTIFICATION DIVISION.
    OBJECT.
    ENVIRONMENTS DIVISION.
        オブジェクトの環境部本体
    DATA DIVISION.
        オブジェクトデータ定義
    PROCEDURE DIVISION.
        オブジェクトメソッド定義
    END OBJECT.

    END CLASS クラス名.
```

図1 オブジェクト指向COBOLのクラス定義の構文

クラス定義では，クラス用のデータ・メソッド，インスタンス用のデータ・メソッドを明確に書き分けることにより，読みやすさの向上を図っている。PROCEDURE DIVISIONでの追加命令文はわずかに2文だけである。

※1) Windows NTは，米国Microsoft Corp.の商標である。

※2) UNIXは、X/Open Company Limitedがライセンスしている米国ならびに他の国における登録商標である。

継承元のクラスを上位クラスと呼ぶ。上位クラスのデータやメソッドの定義が自クラスでも使用できる。複数の上位クラスを持つ多重継承もサポートしている。

(c)　メッセージの受け渡し

オブジェクト指向言語では，モジュール間の呼出し(caller-callee)関係に代わり，オブジェクト間のメッセージ送受信関係が実行の基本である。OOCOBOLではCALL文と類似の構文を用いて実行したいメソッドを持つオブジェクトにメッセージが送信できる。

(d)　メソッドの記述

各クラスで定義するメソッドは，クラスメソッド，インスタンスメソッドに分けて従来のプログラム定義と類似の構文で定義できる。

(e)　インスタンスの生成と管理法

インスタンスの生成機構は，あらゆるクラスの最上位クラス「baseクラス」に“CBL-NEW”メソッドとして用意されている。生成したいクラスに対して“CBL-NEW”を送信するとインスタンスが生成される。

生成されたインスタンスには，識別子(オブジェクト参照と呼ぶ)が付与され，“CBL-NEW”メソッドの実効結果として送信側に返される。以後，このインスタンスは「オブジェクト参照」を使ってアクセスされる。

3　OOCOBOLプログラミング

OOCOBOLはCOBOL85と完全上位互換なので，OOCOBOLによって従来どおりの構造化設計プログラミングができる。OOCOBOLを適用することにより，さらに以下に述べることが実現できる。

3.1　データ抽象の実現

データ抽象の考え方は1970年代中ごろに生まれ，情報隠蔽(ぺい)あるいは部品化と呼ばれてきた。内部処理の変更を外のプログラムに波及させないすぐれた手法である。OOCOBOLでは，個々のオブジェクトの保持するデータへのアクセスをメソッドを介してだけ行うので，そのままでデータ抽象プログラミングができる。

3.2　継承機能による差分コーディングの実現

上位クラスに含まれない自クラス特有の処理だけをコーディングすることを「差分コーディング」と呼ぶ。特にビジネスアプリケーションでは類似の概念を扱うケースが多いので，クラス階層を有効活用することにより，コーディング量の大幅削減が期待できる。

OOCOBOLは多重継承もサポートしているので，強力な差分コーディングができる。

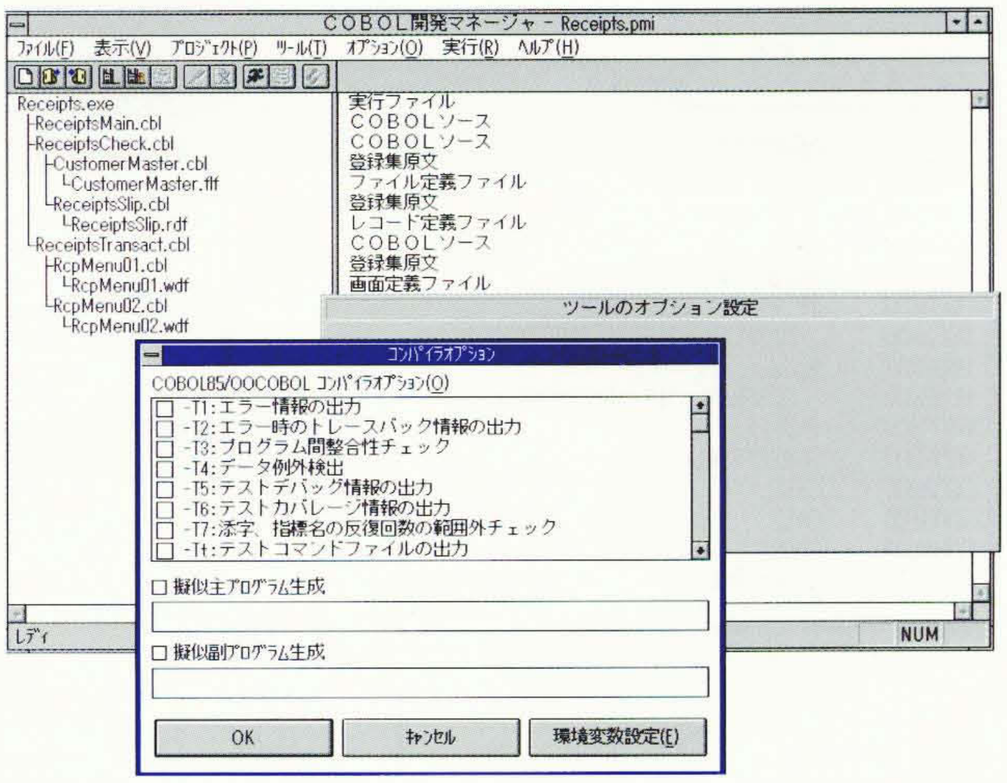


図2　開発マネージャのイメージ画面
開発マネージャからコンパイラをはじめとする各ツールを呼び出すことができる。

3.3　データの動的管理可能

従来のCOBOLは一つのデータ定義で一時に一つの値しか扱えなかった。このため複数件のデータをループで扱う場合など，新データを設定する際に前件データを捨てざるを得ない。例えば集計値の計算を追加する際，前件データ待避などのロジック修正が発生する。

OOCOBOLでは複数件のおのおのに対してオブジェクトが生成・管理できるので，さまざまな要求があとで発生しても新規追加だけで済み，既存部分のロジック変更が不要の場合が多い。

3.4　オブジェクト指向分析・設計手法とのタイアップ

OOCOBOLは従来の構造化手法の中で十分その効果を発揮するうえに，本格的なオブジェクト指向分析・設計手法と連携することにより，さらに効果を上げることができる。

上流工程で表現されるオブジェクトモデルは，プログラム実装工程で作成するオブジェクトとギャップが小さいことから，上流から下流までの開発がスムーズに行える。実世界モデルを反映した構造なので，変更・保守に強いプログラム構造が得られる。

4　OOCOBOLプログラミング環境

OOCOBOLは，UNIX版，Windows NT版をサポートしている。主にNT版に則してOOCOBOLのプログラム環境について以下に述べる。

このプログラミング環境では，OOCOBOL/COBOL85コンパイラ，ブラウザ，テストデバッガ，データベース定義，画面定義，ファイル・レコード定義，ISAM

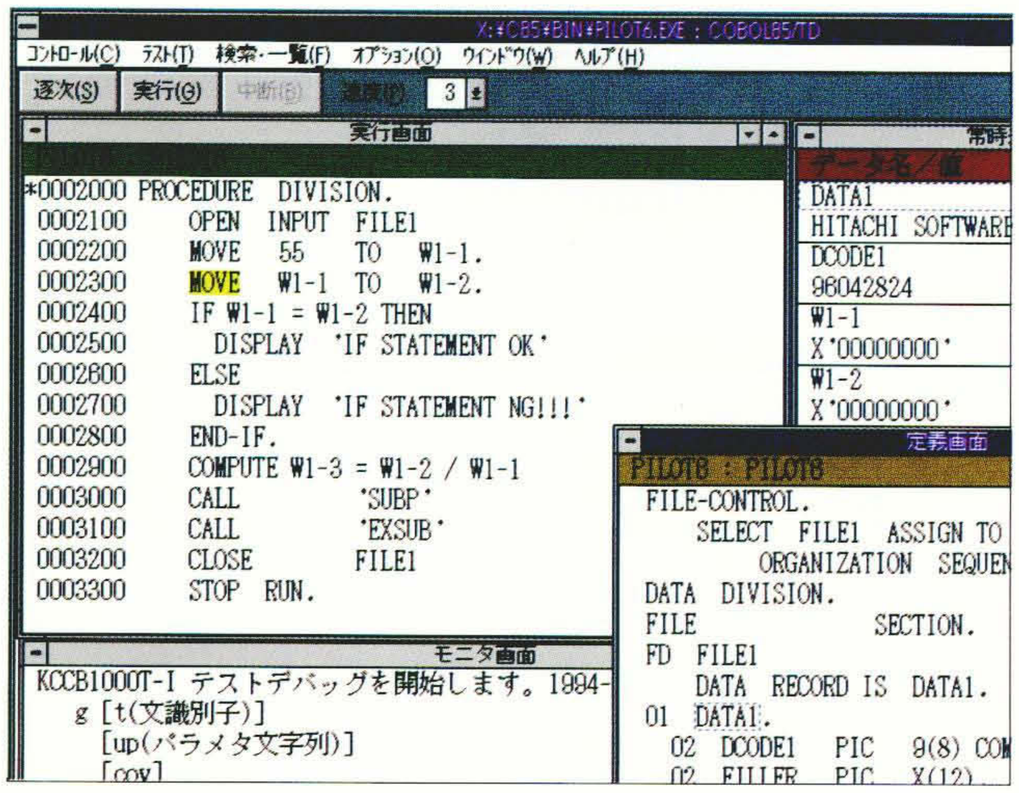


図3 デバッガのイメージ画面

アニメータモードでデバッグを行っている画面である。動作中に実行速度を切り換えたり,一時停止・再実行させたりしてデータの参照・設定などができる。

(Indexed Sequential Access Method), ソート(Sort), およびツール クラス ライブラリが一体となっている。

(1) 開発マネージャ

この環境に内蔵されるさまざまなツールの統合化および業務プログラムの統合的管理を図っているのが開発マネージャである(**図2**参照)。原始文や登録集原文, 画面定義, ファイルレコード定義などのリソース管理, 各リソースに対応したツールの起動制御やツール間の情報交換の制御, およびリソースの依存関係に基づく手順の自動化機能がある。

(2) ブラウザ

クラスやインタフェースの継承関係やプログラム間のCALLの関係をグラフ表示する(**図2**参照)ので, これらの関係がビジュアルに理解できる。また, 表示ボックスをクリックするだけで, そのクラスの持つメソッドやデータの一覧表が表示できる。また, ボックスからエディタを起動して, 該当クラスのプログラムのソースコード編集ができる。

プログラムレビュー時や, 開発時のクロスリファレンスとして使用する。

(3) テスト・デバッガ

アニメータモード, 対話モード, およびバッチモードがある。アニメータモードではプログラムの実際の動きを目で確認しながらテストできる(**図3**参照)。対話モードではコマンドを用いて対話的にテストを行う。バッチモードでは効率よく短時間に大量のテストができる。また, OOCOBOL特有の機能として, メソッドの呼応関係を表示するメッセージトレース機能, および特定メソッドの実行を表示するオブジェクトトレース機能がある。

最適化オブジェクトコード上で動作するので, デバッグ後のリコンパイルが不要である。

(4) 各種定義機能

このほか, データベース定義, ファイル・レコード定義, 対話画面定義などの定義情報をGUI風に定義できる。

(5) ツール クラス ライブラリ

基本クラスとサンプルクラスがある。基本クラスはコレクションクラスなどデータを扱うクラスをライブラリ化したもので, 標準として規格化される見込みが高い。サンプルクラスはマッチング, ソーティングなど業務で頻出する機能を実現するクラスであり, 業務をオブジェクト指向化する際のサンプルとして利用できる。

5 おわりに

ここでは, このたび製品化した, COBOL97の「プリビュー」ともいうべきOOCOBOLについて述べた。OOCOBOLは, COBOL85と完全上位互換であるとともに, オブジェクト指向データベースや分散オブジェクト指向環境などの新しいパラダイムの考え方とも親和性が良い。すなわち, オブジェクト指向か否かという1か0の選択ではなく, 少しずつ適用できるので, 比較的入りにくいといわれるオブジェクト指向の世界へスムーズに入ることができる。

各種ツールを内蔵したオブジェクト指向COBOL・COBOL85向けの本格的な統合開発支援環境をWindows NTおよびUNIXで製品化し, 各種ミドルソフトウェアとも連携を進めている。今, COBOLの世界が徐々にではあるが大きく変わろうとしている。

参考文献

1) Andrew Topper : Object-Oriented Development in COBOL, McGraw-Hill, Inc.(1995)