ソフトウェア工学 108－8
（1996．3．22）

組み込み装置向きオブジェクト指向言語 FORCE－TALK

大山博司† 北出隆彦† 領木正人† 石原敏夫† 直井徹‡

†オークマ株式会社電装事業部
‡岐阜大学工学部電子情報工学科

今日、設計を簡単化するために各種スクリプト言語が開発されているが、これらは汎用コンピュータで動作する小規模なプログラムのためのものである。

筆者らは、組み込み装置にも対応可能かつ十分実用的なアプリケーションが開発でき、また簡単なスクリプティングも可能なスケーラビリティとオブジェクト指向機能、さらにはリモート呼び出し機能を合わせ持つスクリプト言語を目標にFORCE-TALKを開発した。設計にあたって、言語の簡単さを重視すると共に、組み込み装置に適応できる効率と安定性を重視した。

本稿ではFORCE-TALKの概要と実際に開発したアプリケーションを紹介する。

FORCE-TALK : Object Oriented Language for Embedded Systems

Hiroshi Oyama† Takahiko Kitade† Masato Ryoki†
Toshio Ishihara† Toru Naoi‡

†Okuma Corporation
‡Gifu University

Today, many script languages are proposed in order to simplify design. Their purposes are, however, restricted within small applications on general pupose computers.

We propose here a script language, FORCE-TALK, which has simple syntax and provides efficiency and safe execution, so that it can be used to develop applications on embedded computer systems. The language also has scalability. It can be used with small and easy applications to large and more complex applications, with object oriented features, and with remote procedure call facility.

In this article, we introduce an outline of FORCE-TALK and a sample application.

1．はじめに

今日ソフトウェア開発を取り巻く環境は複雑化の一途をたどっている。ソフトウェア開発の問題領域そのものが複雑化しているのに加え、コンピューター技術が複雑化しているためである。

例えば筆者の所属する会社が設計製造するNC装置は、組み込み型のマイコンシステムとしては最も大規模なものの一つであり、１つの製品に含まれるすべてのソフトウェアのソースコードを集めれば、１００万行にも達する程である。これらのソースコードは、C言語やアセンブラにより記述されており、また複数の異なるプロセッサが使用されており、極めて複雑化している。

設計効率を改善し設計品質を向上するには、いかに設計を簡単化するかということが非常に重要である。今日、設計を簡単化するために各種スクリプト言語が開発がなされたり、またコンポーネントウェア[8]と呼ばれるようなビジュアルなプログラミング環境の製品化がなされている。

組み込み装置でも将来的にプロセッサ能力が向上し、より容量の大きなメモリが使用できるような環境が整えば、今日C言語で開発している領域であっても、スクリプト言語によりソフトウェアの設計が可能になると考えられる。

しかし、従来から用いられているスクリプト言語は、本格的なアプリケーション開発には向かない。例えばtcl/tkでは簡単なスクリプトによりウィンドウシステムを活用したアプリケーションを作成することができるが、実際に作成されたアプリケーションは動作が鈍く、組み込み装置での環境には適応できない。

このため筆者らは、組み込み装置の環境でも使用することができ、実用十分なアプリケーションが開発でき、また簡単なスクリプティングも可能なスケーラビィリティとオブジェクト指向機能を合わせ持つスクリプト言語を目標に開発した。

本報告では、この言語を紹介する。

2．設計のポイント

2．1 簡易性

言語の設計で、重視したのは簡易性である。

今日ではコンピュータに関する幅広い技術と、問題領域に対する豊富な知識を兼ね備えることは難しくなり、アプリケーション開発者の負担を減らす意味から、簡単な言語が期待される。

C＋＋言語の教科書[6]は、C言語のものと比べて2.5倍にもなっており、言語機能が肥大化している。組み込み装置のソフトウェア開発の場合、高級言語とアセンブラ言語のインターフェースを必要とすることも多く、言語の機能だけでなく実装にまで立ち入ることとなり、更に負担が増大する。

このような観点から、言語の機能はさほど多くなく、難しくならないように配慮した。勿論実行効率やメモリー効率とアプリケーションの設計効率のトレードオフについても十分に配慮した。

2．2 効率と安定性

組み込み型機器に搭載する場合には、実時間性や限られたメモリー資源のために実行効率、メモリー効率が要求される。また長時間の連続運転でも安定に動作せねばならず、システムダウンは場合によっては多きな損失を発生しかねない。

組み込み型のプログラミングではC言語のmalloc(), free()のような動的なメモリ管理は好まれない。ダングリングポインタやメモリーリーク等の問題が発生した場合、不具合を再現させたり、不具合原因を突き止めるのは、組み込み装置の環境では非常に大変な作業である。それよりは、配列等により静的に領域を確保した方が、不具合も少なく、例えば定義できる最大個数が何個であるか等と言った仕様が明確になり好まれる。

このような背景を考慮して、オブジェクトは基本的に静的に生成することとした。このため構築子や消滅子のように動的な生成消滅を補助する特別なメソッドは用意しない。FORCE-TALKではこのように静的に生成されたインスタンスのこと

をリソースと呼ぶ。

勿論、静的に存在していても多くの場合何らかの初期化を必要とするため、オブジェクトが利用可能となるタイミングに関して完全とは言えないが、少なくとも生成するための手続きを省略できる分、メモリは節約できる。

2．3 メモリ管理

前項で述べたように、基本的にはオブジェクトを静的に生成することとしたが、文字列長や配列の大きさが固定されてしまうのは、プログラムの融通性を著しく損なうことになるため、文字列や配列に関しては動的な生成／消滅／サイズ変更を行なうこととした。これに伴い参照数による寿命管理機構を設けた。

また、このメモリ管理機構を用いてオブジェクトの寿命管理も可能とした。ただし、オブジェクトは基本的には静的生成であり、動的な生成はコピーによってのみ可能とした。コピーにより生成されたオブジェクトは動的リソースと呼ぶ。

オブジェクトの寿命管理は、C言語で誤りを起こしやすい点の一つなため、寿命管理が自動的に行なわれることは効果がある。FORCE-TALK では動的に生成されたもののみが対象となるため、寿命管理は極めて軽いものとなっており、組み込み装置に向いている。ガベジコレクション機能も実装しているが、通常の利用の仕方ではガベジコレクションは発生しない。

2．4 移植性

組み込み型機器でも今日 RISC 系のプロセッサが普及し始めている。今後更に高性能化が見込まれることから、移植性は十分に考慮しなくてはならない。前に述べたとおり、効率も重視したいことから、C言語へのプリプロセッサとしてコンパイラを実装することとした。

2．5 型チェック

組み込み装置用として、高い効率を要求されること、また、実行時よりはなるべくコンパイル時に不具合を検出したほうが、テスト前に少しでも不具合を排除できることから、静的な型チェックを行なうこととした。

2．6 継承

継承は、非常に強力な機能であるが、プログラミング効率のみからむやみに継承が用いられれば、全体としてはかえって分かりにくいソフトウェアになるなどの問題もある[7]。また、同種のしかし多少性質の異なるクラスが存在しない場合には、継承は役に立たない。実際 NC 装置のような組み込み型の装置では、インスタンスの数は極めて多いが構造が単純であるか、かなり複雑だがインスタンスは少なく一つかせいぜい数個で、継承によるプログラムの再利用が期待できる場所は比較的少ないと思われる。

それよりは、巨大なソフトウェアをいかに独立性の高いサブシステムに分割するか、それらの内部構造をいかに覆い隠し利便性の高いインターフェースを与えるかが、オブジェクト指向の本質であると考える。

FORCE-TALK では継承と言うよりはインスタンスの特化と言うべき機能を用意している。この機能は、リソースメソッドと呼ぶ特定のリソースにのみ適応するメソッドを記述することで実現している。

2．7 プロジェクトと階層化制約

FORCE-TALK では、複数のクラスをまとめて１つのプロジェクトを形成する。プロジェクトは、相互に参照し合うクラスの集まりであると同時に、コンパイルを行なう単位である。

プロジェクトには、ＩＤとして０～２５５の間の整数を与える必要がある。このＩＤが大きい方から小さい方を参照（import:)することは許すが、

この反対は禁止している。これを階層化制約と呼んでいる。

これにより、二つのプロジェクトの間に参照関係が存在する場合、必ず一方から他方の参照に限られる。規模の大きなアプリケーションでは複数のプロジェクトに分割することになるが、この場合アプリケーションの内部においても階層化への対応を要請するものであり、大域的な循環依存状態が発生しないことが保証される。

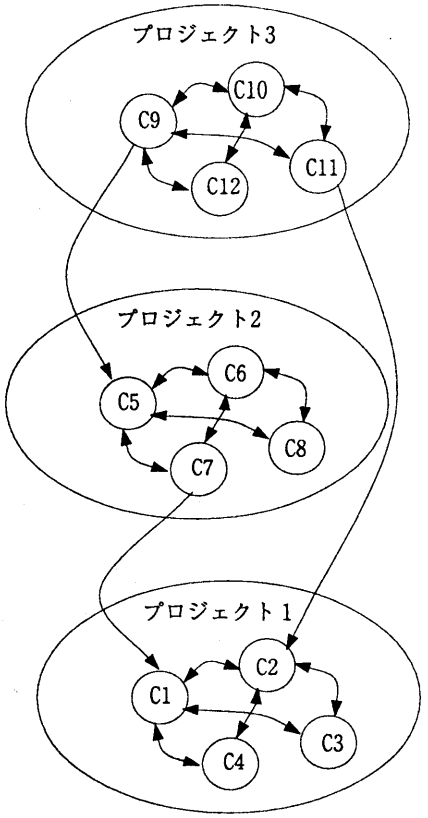

図１階層化制約

図１は、階層化制約の概念図であり、クラスＣ１～Ｃ１２は、それぞれのプロジェクト内では相互に依存し合うが、他のプロジェクトにまたがっては、一方向にしか依存しない（呼出しが行なわれない）ことを現す。

２．８ ＩＤ

メソッド、関数、リソースにはＩＤが付けられる。このＩＤは、プロジェクトＩＤと、各プロジェクトあるいはクラス定義ファイルでの出現順番とからなる３２ビットの整数である。FORCE-TALK ではこのＩＤによりリンクを行なう。

従って、名前とリンク時の結合とは関係がなく、日本語以外の他の言語で作成されたプログラムであっても、クラス定義ファイルの出現順序が対応していれば、リンクすることが可能になる。クラス定義ファイルを他の言語に書き換えることは、簡単な構文解釈と文字列のマッチングにより可能である。現時点ではこの変換機能はサポートしていない。

２．９ ＲＰＣ

ＲＰＣ(Remote Procedure Call)は、あるプロセッサで動作するプログラムが別のプロセッサで動作可能なプログラムを呼び出すことである。

先にも述べた通り、この言語はリンケージにおいて名前ではなく、ＩＤとして３２ビットの数値を割り当てていること、階層化制約によりＩＤの高い方から低い方への呼び出しに限られていることから、簡単にクライアントとサーバーを分けることができ、言語仕様の範囲内でリモート呼び出しに対応できる。通常 import: により参照していた下位のプロジェクトを remote: に変更するだけでよい。これは、import: で参照されている場合には、呼び出しに伴う効率をあげるためＩＤを名前としてリンク時に参照を解決していること、ＲＰＣを介した呼出しには若干の差異があるため、明示する意味を含んでいる。

図２は、先程の図１で示した構造のプロジェクト３をクライアントにプロジェクト１，２をサーバーに分離した時の状態を現している。この図から分かるようにプロジェクト３からプロジェクト１，２への呼出しに限られるため明確にクライア

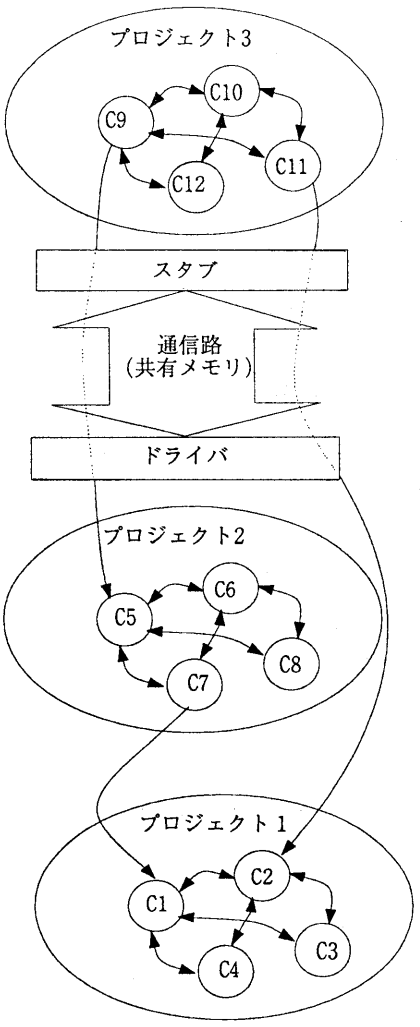

図２ＲＰＣ

ントとサーバーを分けることができる。

ＲＰＣと通常の呼出しの差異は、引数や戻り値の配列が通常の呼び出しとは異なるコピー規則となること、リソースの所在とメソッドの所在に関する注意が必要となることである。

FORCE-TALKの例外処理（Ｃ＋＋の throw-catch に相当 FORCE-TALK では trap-exception)もＲＰＣを透過して例外を伝えるようになっており、単純なデータ型を引数や戻り値に使用している場合は、リモート呼び出しを全く気にする必要が無い。

ＲＰＣ機能はマルチプロセッサ環境で、プロセッサ間の機能呼び出しに用いる目的で導入した。一般的な組み込み装置では共有メモリを通信路として使用するが、一般的なネットワークなどに置き換えることは勿論可能である。

Ｃ言語でのＲＰＣとしては例えば SUN RPC [9] があるが、これでは専用のＲＰＣコンパイラでコンパイルする必要があり、言語もＣ言語によく似たＲＰＣ専用の言語を使用してＲＰＣによる呼出しを定義する必要があるが、FORCE-TALK ではＲＰＣに対応できるだけの情報が始めから整っているため、特別な言語やコンパイラを必要としない。

２．１０ 構文

ソースプログラム自体のドキュメント性を高めることも配慮した。具体的には日本語を使用し、構文も日本語の文書に近くなるようにした。これはプログラムの読みやすさを向上させるものである。

次に示す文は、メソッドの定義文であり、宣言文であり、呼び出し文の例である。FORCE-TALK ではこれらを一致した形式とした。(++ は行の継続を現す)

```
( x%, y%, w%, h% ) =                          ++
        gp@( GRAFPORT ) の位置大きさ
gp@( GRAFPORT ) の位置大きさを                ++
        ( x%, y%, w%, h% ) に変更
```

この形式は smalltalk などのように分離したメソッド名と、変数に記号を付けて型を区別することにより実現しているもので、主語でも目的語でも先頭に来る日本語には特に適した形式であると思われる。なお x%, .. h% が整数を格納する為の変数であり gp@ はレシーバとなるオブジェクトで、続く括弧の内部はクラス指定であり、呼び出し文では省略が可能である。実際、この形式を用いることによりプログラム自身がコメントのよう

になり、読みやすさは向上する。記述量は多くなるが、テキストエディタの機能の向上によりコピー／ペーストを使うのが一般化していること、宣言文と呼び出し文の形式が一致していることから面倒ではない。しかし、このような構文を導入しても、なおif文の条件部には、変数と変数または定数の比較が多くif文付近のコメントが減少していない。

2．11 関数

クラスメソッドに相当するものはなく、代わりに関数を使用できるようにした。例えば

GRAFPORT 初期化

はGRAFPORTクラスの初期化を行う関数である。GRAFPORTをレシーバーとして「初期化」というメッセージを送る形式になっているが、実際には分離した1つの関数名となっている。関数を導入したことにより

```
y# = sin b#
a# = atan( y# / x# )
```

のように数学的な式の記述が、より自然な形で行える。

ボディー変数(インスタンス変数のこと)はすべて非公開であるため、関数では直接これらを扱うことができず、構造的なデータを扱うにはクラスを設け、メソッドを定義しなくてはならない。これにより、必要以上に、非オブジェクト指向の（カプセル化されていない）プログラミングが行なわれないようにしている。

3．簡単なプログラムの例

FORCE-TALKによる簡単なプログラムの例を示す。この例はX-Windowハンドブック[3]の付録に掲載されているX-Window版Hello Worldを少し簡略化してFORCE-TALKで記述した例である。FORCE-TALKのプログラムを記述する際に必要となる3つのファイルの全リストを示す。

1）プロジェクト定義ファイル

プロジェクト定義ファイルはＩＤとimport:するプロジェクトとクラスを記す。

```
projectName:   HELLO
projectID:     239
import:
        STDLIB
        TOOLKIT
class:
        MAIN
end:
```

2）クラス定義ファイル

クラス定義ファイルには、公開するメソッド、関数、リソースの宣言及びボディー変数とそのデフォルトの初期値を記す（この例では関数のみ記している）。

```
class:
   MAIN
function:
   res& = main ( argc%, argv$[] )
end:
```

3）プログラムファイル

プログラムファイルは、少なくとも公開したメソッド等の定義を記述する。FORCE-TALKで記述したこの版はTOOLKITを利用しているため若干プログラムが長くなっている。リソースhello_panel@(PANEL)の定義部分で、マウスボタンの押下イベントに対するアクションがリソースメソッドとして記述されており、元々PANELクラスに定義されているマウスボタン押下に対応するメソッドはオーバライドされる。

実際このプログラムを作成する場合には、後で紹介する画面エディタでプログラミングを行ない、このリソースメソッドの部分のみを書き加えるだけでよく、他の部分はこのツールが自動生成

する。

```
function:
  res& = main( argc%, argv$[] )
script:
  ウィンドウサーバー初期化
  ツールキット初期化

  _hello_eventer@ を登録
  _hello_eventer@ をフォアグラウンド化
  _hello_eventer@ 処理実行
end:

resource:
  _hello_eventer@( EVENTER )
body:
  _hello_frame_0@
  ,
  {}( FKFRAME )
  {}( DATAIN )
  rootwin@
end:

resource:
  _hello_frame_0@( FRAME )
body:
  #FRAME_MAIN
  #TRUE
  {
    { hello_panel@ }
  } -- array of PANELGROUP
  ,
  {}
end:

resource:
  gp@( GRAFPORT )
body:
    {
     0, 80, 640, 320
     1, #WIN_NOT_RETAINED
     white@( COLOR )
     black@( COLOR )
    }( WINDOW )
    , , ,
    white@( COLOR )
    black@( COLOR )
end:

resource:
  hello_panel@( PANEL )
body:
      -- nothing on panel
method:
 pass%=これ@(PANEL)でマウスが押下された++
                    ev@( EV_BTN_PRES )
var:
  x%, y%
script:
  ( x%, y% ) = ev@ 発生位置
  gp@の( x%, y% )に文字列 "hello world"を描画
end:
```

4．アプリケーション

次に FORCE-TALK で開発したアプリケーションを示す。図3は、画面エディタと呼んでいるツールで FORCE-TALK で開発したアプリケーションであり FORCE-TALK の実用性を試すための実験台としても使用した。

このツールは、丁度ドローツールのように、ウィンドウ上にウィジェット（部品）を配置し、ウィジェット毎にイベントに対応したスクリプトを記述することができるもので、最終的には FORCE-TALK のソースプログラムを出力する。

このツールで特筆すべきことは、編集データの入出力のためのプログラムがほんの数行だけだということである。FORCE-TALK では基本的にリソースを静的に生成するため、画面エディタを構成するリソースは全て静的に生成される。一方編集により画面上に配置されたウィジェットは動的に生成されるため、リソースのディープコピーで動的に生成されたもののみコピーを行なうことにより、状態を保存することができる。従って、状態を保存するために各クラスに特別なメソッドを用意する必要もないし、メモリ全体をダンプすることによって、僅かな編集結果が巨大なファイルになることもない。

画面エディタと共に開発した TOOLKIT は、ポップアップメニューの現場向き代替機能として開発したポップアップファンクション機能や、テキストエディタを含む数種類のウィジェットを備えており、マルチスレッド（リアルタイムＯＳ上ではマルチタスク）にも対応している。もちろん

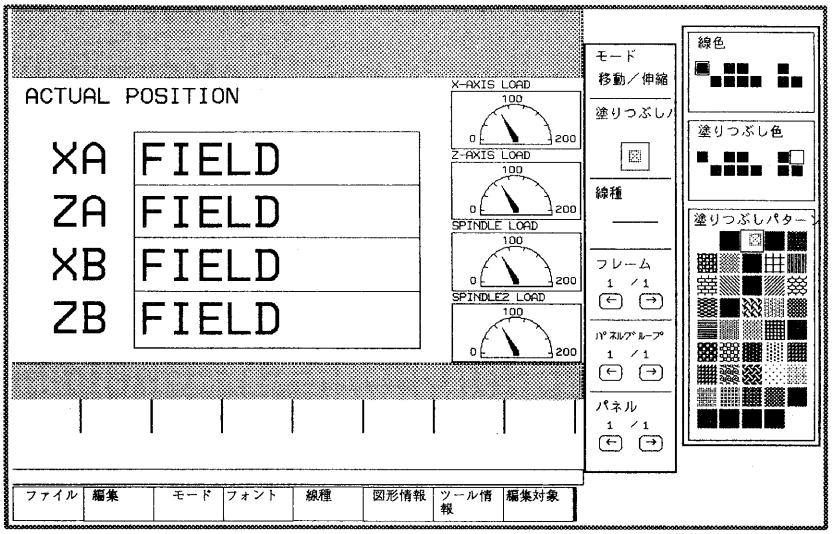

図3 画面エディタ

TOOLKITもFORCE-TALKにより記述されている。画面エディタはFORCE-TALKの実験台としてばかりでなくTOOLKITの実験台としても使用した。

5．おわりに

以上紹介したFORCE-TALKは、組み込み用のRISCプロセッサ上で実行効率、処理効率ともに満足の行く性能が得られている。組み込み装置の環境でも使用することができ、実用十分なアプリケーションを開発できるという目標はかなり高いレベルで達成された。今日ではFORCE-TALK、画面エディタ共にユーザー数も増加しており、社内ツールとして十分な効果をあげている。

現在はコンパイラのみとなっており、組み込み装置のユーザーインターフェース部の適用に限られているため、今後スクリプトのインタープリタを開発し、より簡単な利用を実現すること、逆にオーバーヘッドを最小化しリアルタイムな領域にも適応できる、スケーラビリティのあるスクリプト言語を実現したいと考えている。また、未サポートである外国語への変換機能も開発する予定である。

参考文献

[1]梅村恭司: Smalltalk-80入門 サイエンス社 1986
[2]梅村恭司: 考える道具としてのMacintosh/HyperCard 共立出版 1989
[3]Oliver Jones著 西村 亨監訳: X Windowハンドブック アスキー出版局 1990
[4]BERTRAND MEYER著 二木厚吉監訳: オブジェクト指向入門 アスキー出版局 1990
[5]中川 正樹: 母国語プログラミングへの方式，実践とその効果 ソフトウェア工学研究会 85-2 1992
[6]B.ストラウストラップ著 斉藤信男他訳: プログラミング言語C++第2版 アジソンウェスレイ・トッパン 1993
[7]B.F.ウェブスター著 細井拓史訳: オブジェクト指向開発の落とし穴 1995
[8]星野友彦: コンポーネントウェアが来た 日経コンピュータ (No.371) 1995
[9]SUN OS 4.1.x マニュアル Network Programming Guide Sun 1990
[10]宮田重明　芳賀敏彦: Tcl/Tk プログラミング入門 1995