マルチメディア通信と分散処理 75－5
（1996. 2. 29）

分散処理用クラスライブラリによるプログラム開発環境の分散化

鈴木寿郎* 水越剛成* 中澤修**

*(株) 沖テクノシステムズラボラトリ **(株) 沖電気工業

マルチメディア処理用組込プロセッサ mmX のプログラムクロス開発環境は，Tcl/Tk によるスクリプトによってその上位部分が作られている。複数のホストマシンを用いた分散的な開発を支援するため，C++ のための分散処理用クラスライブラリ DOL/C++ で拡張した Tcl/Tk をこの上位部分へ適用することによって，プログラム開発環境を分散化した。本稿は，このプログラム開発環境，クラスライブラリ，Tcl/Tk 拡張，および分散化された開発環境について述べる。

The Construction of a Distributed Development Environment Based on the DOL/C++ Class Library

Hisao Suzuki* Takanari Mizukoshi* Osamu Nakazawa**

*Oki Technosystems Laboratory **Oki Electric Industry

We have built the upper-level components of a cross-development environment for mmX, an embedded processor for multimedia applications. Most of the components are written in Tcl/Tk. In order to make the environment usable as a distributed one, we extend the Tcl/Tk interpreter with the DOL/C++ class library, which aims at distributed processing, and apply it to the components to enhance the facilities.

In the present paper, we describe the development environment, the class library, the extension, and the enhanced facilities.

1 はじめに

著者らはマルチメディア処理用組込プロセッサ mmX (MultiMedia eXecutor) [1] のためのプログラム開発環境の上位部分として，Tcl/Tk [2] によるツール群を作成した。本文では，複数のワークステーションを開発ホスト・マシンとする協同作業を実現するための，ソフトウェア開発環境の分散化について述べる。

まず，現在の開発環境を概説し，分散化への問題点と解決法を述べる。次いで分散化のための C++ クラス・ライブラリと，その Tcl/Tk へのマッピングについて記述する。最後にその拡張による開発環境の分散化機能について述べる。

2 mmX 開発環境

mmX のプログラム開発環境は，ホスト・マシンであるワークステーションから，mmX ターゲット・マシンまたはそのシミュレータ・プログラムを制御する，クロス開発環境として作られている。

ホスト・マシン上の開発環境の上位はワークベンチ，デバッガ，プログラム・ビルダの各ツールから構成されている。これらはそれぞれ Tcl/Tk のスクリプトとして作られており，Tcl/Tk による GUI を提供する (図 1)。

デバッガとプログラム・ビルダは単体でも利用可能であるが，ワークベンチを中心にしてこれらを連繋させることもできる:

プログラム・ビルダ ⇌ ワークベンチ ⇌ デバッガ.

mmX ターゲット・マシンやそのシミュレータを制御する低水準の C/C++ プログラムは，Tcl/Tk の拡張コマンドとして実装されている。これはデバッガのスクリプトから Tcl/Tk インタープリタを介して操作される。

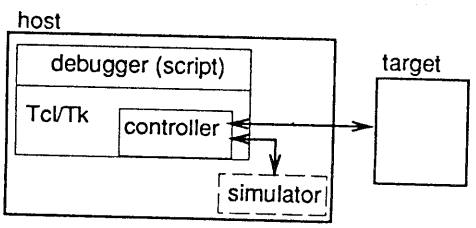


図 2: ターゲットの制御

以下に，開発環境を構成する各ツールについて簡単に説明する。

2.1 プログラム・ビルダ

プログラム・ビルダは，プログラムのコンパイル等に使用する。コンパイル，make ファイルの設定と生成，make ファイルのブラウズなどの機能を持つ。

2.2 デバッガ

ターゲット用の実行プログラムの実行やデバグを行う。C 言語レベルのデバグとアセンブリ言語レベルのデバグの 2 種類のインタフェースが用意されている。

2.3 ワークベンチ

ワークベンチは，プログラム・ビルダとデバッガ，および emacs などのエディタを下位ツールとする統合環境を提供する。ワークベンチには以下の機能がある。

- ファイルの管理
  ソース・ファイル，make ファイルおよび実行ファイルを管理する。ファイル管理は，ファイル・リストへのファイル名の登録と，そのファイル名をパラメタとしての下位ツールの実行という形で行われる。ソース・ファイルに対応する下位ツールはエディタ，make ファイルに対応する下位ツールはプログラム・ビルダ，実行ファイルに対応する下位ツールはデバッガである。
- 各ツールの管理と連繋
  デバッガやエディタなどの各ツールを連繋させることで，各ツールを単独で使う場合より効率良く作業できる環境を提供する。この連繋機能には次のものがある。
  1. コンパイル後，更新した実行ファイルをターゲット・ファイルのリストに登録する。
  2. プログラム・ビルダで作成した make ファイルを make ファイルのリストに登録する。
  3. デバッガでデバグ中に実行ファイルがコンパイルされて更新された場合，変更されたことをデバッガへ通知する。
- 各ツールのカスタマイズの支援
  各ツールのコンフィギュレーション・ファイルの変更と作成を支援する。

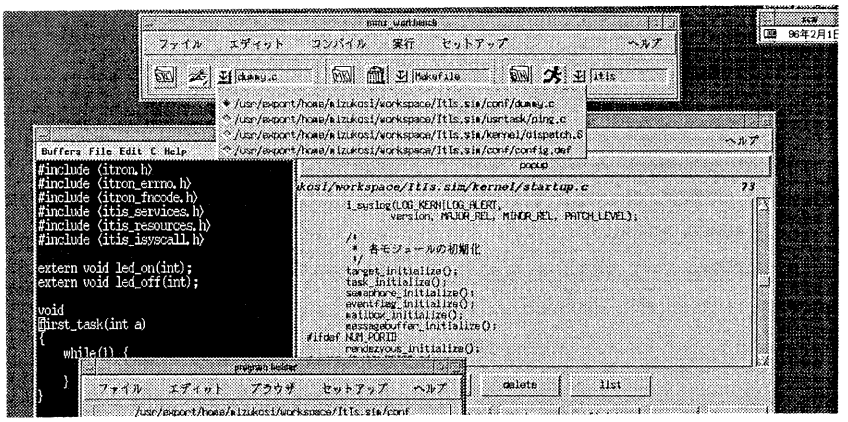


図 1: mmX 開発環境

3 分散化への問題点と解決法

複数台のワークステーションをそれぞれ開発ホスト・マシンとして複数の者が協調して作業するために，開発環境を分散化させることを考えるとき，次のような問題がある。

3.1 問題点

開発環境の各ツールのプロセス間通信は，Tcl/Tk の send コマンドで実現されている。組み込みコマンドのひとつである send コマンドは，X11 Window System のディスプレイを共有する他のインタープリタのプロセスに，Tcl/Tk プログラムを内容とする文字列を X11 のプロパティを介して送信する。受信側はプログラムをイベントとして受け取り，イベント処理の一部として評価する。送信先としての各プロセスの名前も X11 のプロパティを介して参照される。

$$interp_1 \overset{\text{program}}{\underset{\text{result}}{\rightleftarrows}} interp_2$$

send コマンドの提供する通信形態の特徴は，通信相手の検索がプロセスごとの名前に基づいていることと，一度に一対一ずつの通信をサポートすることである。これは，ひとつのツールから起動されて作動する数個のプロセス間の通信には，十分な仕様といってよい。また，プログラムが受信側で無条件に実行されることは，例えばスクリプトを対象とするデバッガの実現にとっては最適である。

しかし，これは，通信しあうプロセスが同一のディスプレイを共有しなければならないため，分散化には適用できない。ホスト間にまたがる通信機能を用意する必要がある。

また，各ツールの柔軟な運用性，例えば非同期的な起動と終了を支障なく実現するためには，send コマンドのような通信形態に加えて，プロセスに対する名前またはグループの緩やかな関係付けや，特定のグループへの同報通信を行う機能が望まれる。一方，受信したプログラムの実行については，安全上，その機能を簡単に実装できるだけの枠組を用意するにとどめるのが好ましい。

3.2 解決法

著者らは，開発環境の分散化を可能にするため，以前より開発した DOL/C++ [3, 4] の提供する分散処理用 C++ クラス・ライブラリを用いることにした。次節で述べるように，このクラス・ライブラリは必要な機能をほぼ提供している。

分散化は次のように行われる。まず，通信機能をスクリプトから利用可能にするため，DOL/C++ クラス・ライブラリを用いた C++ プログラムで Tcl/Tk 拡張コマンドを作成する。つぎに，この拡張コマンド (および mmX 制御用コマンド) を組み込んだ拡張 Tcl/Tk インタープリタを作る。最後に，この拡張コマンドを用いて，開発環境の各ツールのスクリプトに必要な改訂と追加を施す。

4 DOL/C++ の概要

分散化に用いている第2版 DOL/C++ について概説する。DOL/C++ は，プロセス間通信での読み書きを行う通信端点を，通信者 (correspondent, corr.) という概念を用いて抽象化する。各通信者オブジェクトはそれぞれひとつのプロセスに属しており，そのプロセスの通信端点としてはたらく。

基本的な通信は，通信者オブジェクトから通信者オブジェクトへのメッセージの非同期的な伝達として表される。

$$corr_1 \longrightarrow corr_2$$

同報通信には通信者と基底クラスを同じくする伝達者 (messenger) を使用する。メッセージを伝達者へ送信するとき，ひとつの伝達者が複数の通信者へマップされ得る。

$$\begin{aligned} corr_1 \longrightarrow messenger &\longrightarrow corr_2 \\ &\longrightarrow corr_3 \\ &\longrightarrow corr_4 \end{aligned}$$

伝達者にはそれぞれひとつの名前を付けることができる。任意の値の文字列を名前として用いることができる。伝達者の名前と伝達者値との対応は一対一であり，各プロセスはこの名前で伝達者を参照する。

通信者もまた伝達者とは別の名前空間で名前を付けられる。通信者の名前は，他のプロセスが名前で通信者を参照するためのものであると同時に，一種の排他的な資源であり，名前の権利を動的に獲得，譲渡ないし放棄すること，ひとつの通信者が複数の名前を獲得することが可能である。

表 1: 通信者名の操作

獲得	corr.acquires(name, seconds) corr.acquiresSecretly(name, seconds)†
公開	corr.announces(name)
放棄	corr.renounces(name)
譲渡	corr.transfers(name, another_corr)

seconds は操作に許される時間 [秒] の上限

†公開するまで他からの名前参照不可能

実際の通信は，2種類の通信サーバを用い，ソケット・ライブラリ [5] によって実装されている。通信サーバ POB (post-office box) は，各ホストの各ユーザごとに設けられ，主にそれぞれのメッセージ私書箱としてはたらく。通信サーバ PM (postmaster) はシステム全体でひとつであり，すべての POB をたばね，通信者と伝達者それぞれの仮想的な名前空間を維持する (図3)。

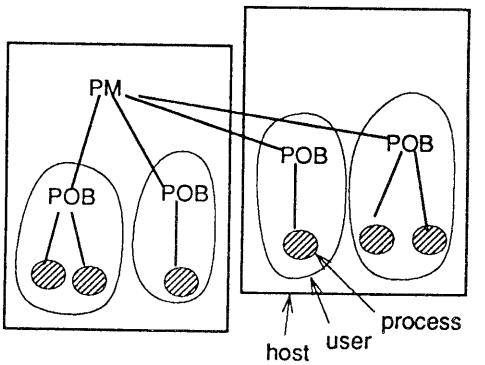


図 3: DOL/C++ の通信サーバ系

通信者から通信者への一対一通信では，受信側の POB が介在するだけであり，PM に負荷はかからない。ソケットと通信者オブジェクトとの関係は間接的であり，ひとつのプロセスは資源をほとんど費やすことなく 100 個以上の通信者を設けることができる。

また，通信者には公開または非公開の属性を任意に設けることができる。公開属性は他のプロセスからキーにより属性値の参照と書換えが可能である。これは内部的には通信者と通信者の通信で実装されている。

例えば整数の公開属性を，ある初期値を指定して corr に設けるには，次のようにする。

```
d_Prop<int>* prop =
    new d_Prop<int>(KEY, 6102);
corr.putsProperty(prop);
```

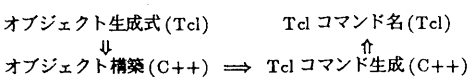
また，別のプロセスからその値を参照して int 変数 value に格納するには，次のようにする。

```
d_Access access(remote_corr, KEY);
access >> value;
```

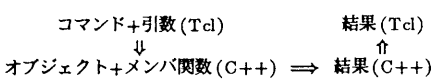
公開属性の参照と書換えの手続きは，属性クラスの仮想関数の上書きによって定義できるから，オブジェクト・メソッドの同期的なリモート呼び出しとしても使用できる。

5 Tcl/Tk の拡張

このクラス・ライブラリによる Tcl/Tk の拡張の手法として，次の図式のように，C++ クラスのオブジェクトを Tcl コマンドへ写す方法を採った。



生成される Tcl コマンドは C++ オブジェクトと一対一に対応する。そのコマンドの実装プログラムの中では，Tcl コマンドの引数に基づき C++ メンバ関数への分岐が行われる。したがって，典型的には次のような図式で処理が行われる。



例えば Tcl スクリプトで `d_create` コマンドを実行すると，そのプロセスに属する通信者オブジェクト *corr* が内部に構築され，その値の文字列表現である *corr*`.image()` の結果をコマンド名とする新しい Tcl コマンドが生成される。そしてそのコマンド名が実行結果の値として返される。

したがって，典型的には，生成されたコマンド名をスクリプト内で後から参照できるようにするため，`d_create` を実行するときは次のように結果の値を変数に格納する。

```
set c [d_create]
```

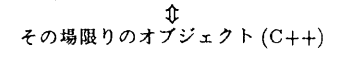
ここで，通信者名を獲得するために，

```
$c acquire -announce "foo/Corr0" 10
```

とすると，このコマンドを実装しているプログラムの中で *corr*`.acquires("foo/Corr0", 10)` が呼び出される。この C++ メンバ関数で名前の獲得に失敗したときは，その失敗は Tcl の例外状態に翻訳され，スクリプトでの `catch` コマンドによる例外処理にゆだねられる。

Tcl コマンドへ写される C++ オブジェクトには，通信者のほかに，公開属性へのアクセス・クラス `d_Access` のオブジェクトがある。

遠隔プロセスの通信者を表すオブジェクトなど，基本的に他のオブジェクトのメンバ関数の引数としてだけ使われ，適当な値の組から低コストで再構成可能なクラスは，元の値への逆変換が可能な文字列へ写されることで Tcl の値となる。コマンド実装 C++ プログラムは，コマンド引数として与えられたこのような文字列値から，その場限りの一時的なオブジェクトを再構成する。



この Tcl/Tk 拡張のための C++ プログラムの大きさは 1000 行強である。このプログラムでは，新設の C++ クラスとして，受信ハンドラおよび公開属性機能の実装のために，二つの通信者属性クラスを派生させている。以下にこの拡張 Tcl/Tk の若干の使用例を挙げる。

```
## 受信ハンドラの文脈では [d_msg 引数] で
## メッセージの内容が得られる。
proc a_handler {} {
   set text [d_msg text]
   puts $text
}

## "グループ" は "伝達者" で実装され，
## 名前で参照される。
set group1 [d_group "foo/Group1"]

## 通信者を生成し，ハンドラを設定する。
set corr [d_create]
$corr handler a_handler

## グループあてに，あるリストを同報送信する。
$corr send $group1 [list $a $b $c]
```

6 ツールの分散化

分散化は，各ホストにおいてそのディスプレイ上のツールを統合しているワークベンチを中心に行った。

現在，分散化のために組み入れられた機能は

- NFS でファイルを共有している場合に対処するため，現在，対象としているファイルを，相互に，他のワークベンチのファイル・リストの中に入れて表示可能とした。
- テキストによる簡便な対話を可能にするため，メッセージ・ボードのウィンドウを設けた。どのメッセージ・ボードにも次のテキストが同時に表示される。
  - 人間がキーボード等で入力した文

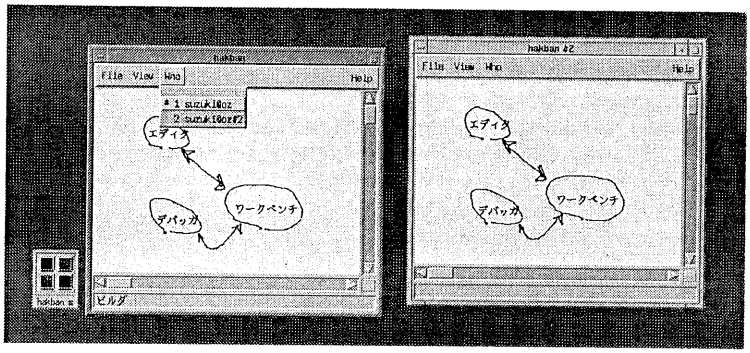


図 4: 分散白板

- プログラム・ビルダからのコンパイル終了通知
- ワークベンチの起動および終了の通知

- 簡単な図による情報の交換を可能にするため，分散白板(図4)を設けた。マウスで描いた線図，および任意の位置に書き込んだ文字列が，どの白板にも同時に表示される。

これらは，基本的に DOL/C++ の伝達者による同報通信で実現されている。各ワークベンチは，任意の時点で起動または終了することができる。

分散白板については，過去の描画履歴それ自体が共有すべき状態を構成している。途中から参加して来た白板が今までの履歴データなどを参照するために，状態保持用サーバが設けられている。このサーバは，それ自体 Tcl/Tk スクリプトであり，2重起動を防ぐために，通信者の名前の排他性を利用すると同時に，各種データのアクセスに通信者の公開属性を利用している。白板は次のように，通信者名の参照の成否によって，状態保持用サーバを自動起動する。

```
if [catch {set server [d_corr $sname]}] {
   exec [hb_serverCmd] $version &
   set server [d_corr $sname 30]
}
set accssToLP [d_access $server LastPOINT]
```

6.1 未実装機能

現在，以下の点が未実装である。

- ターゲット・ファイルのコンパイル終了などは，各ワークベンチのメッセージ・ボードに広報されるが，ファイルの因果関係に基づいて遠隔ツールを自動連動させると便利である。
- デバッガ画面など，他者と GUI の表示を共有できると便利なことが多い。Shaped Object [6] のような機能を組み入れた mega-widget [7] を作成して Tcl/Tk の widget と置き換えれば，ほとんど書き変えることなく既存のスクリプトにこの機能を組み入れることができる。

これらは簡単なスクリプト上で実験されているが，まだまとまった形では実装されていない。

7 おわりに

ここでは DOL/C++ クラス・ライブラリによる mmX プログラム開発環境の分散化について概説した。

現在，mmX 上に μITRON3.0 が実装されつつある。著者らは μITRON 用デバッガの作成とその分散化とともに，未実装部分の実現および各スクリプトの全体的なレビューと改訂を行っている。

参考文献

[1] 河合 他: マルチメディア用 RISC コントローラとその応用, 情処研報 ARC 110-21, 1995.

[2] Ousterhout, J. K.: *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[3] 鈴木，中澤，佐藤: 分散ソフトウェア開発用ツールキット—ライブラリ—, 情処 48 全大, 6D-6, 1994.

[4] 鈴木，中澤: 分散ソフトウェア構築のための DOL/C++ クラスライブラリ, 情処研報 DPS 66-5, 1994.

[5] UNIX Software Operation: *UNIX SYSTEM V RELEASE 4 Programmer's Guide: Networking Interfaces*, Prentice-Hall, 1990.

[6] 横田実: Shaped Object による情報の分散共有, 情処研報 DPS 73-13, 1995.

[7] Jaeger, S.: Mega-widgets in Tcl/Tk: Evaluation and Analysis, *Tcl/Tk Workshop*, USENIX, 1995.