Vol. 28 No. 6 情報処理学会論文誌 June 1987

実時間オペレーティングシステム R²-86 の開発とその移植性†

大久保 英嗣†† 津田 孝夫†† 楠田 修三††
小林 正典†† 杉村 邦彦†††
白濱 和人††† 友田 和伸†††

R² は，ロボットマニピュレータや NC 工作機械等のロボットシステムにおける各種制御装置に組込み可能な実時間オペレーティングシステムである．R² は，従来の制御用ソフトウェアにおける設計，開発および保守時の問題点を解決するために開発された．すなわち，R² は制御用ソフトウェアの応答性のみならず移植性の向上を主目標としている．さらに，アプリケーションプログラムの開発を容易にするためのユーザ親和性を持ったインタフェースを実現するよう設計されている．R² は，現在，マイクロプロセッサ 8086 上に構築されている．我々は，これを R²-86 と呼んでいる．R²-86 の実現は，主にC言語を使用して行われている．将来的には，マイクロプロセッサ 68000 をもサポートする予定である．本論文では，R²-86 第1版の開発の背景，設計目標および全体の構成について述べる．さらに，R²-86 における移植性向上策であるプログラムの機械依存部の記述方式とロボットプリミティブについて述べる．

1. はじめに

今日，ファクトリオートメーションの分野では，ソフトウェアの移植性と保守性が非常に大きな問題となっている[1)~5)]．これは，制御用ソフトウェアが，ロボットマニピュレータや NC 工作機械等の個々の機械に関して個別に開発されていることに起因する．この問題を解決するために，我々は R² と呼ばれる移植性の高い実時間オペレーティングシステム（以下 OS と記す）を開発している[6)~10)]．R² は，上記の両方の型のロボットシステムを初めとする各種制御装置に組込み可能な構成となっている．R² は，現在，マイクロプロセッサ 8086 上に構築されており，これを我々は R²-86 と呼んでいる．R² は，制御用ソフトウェアが使用される種々の環境に対応して再構成可能な OS である．そのため，システムのモジュール化を可能な限り推進することによって，ロボットシステムのハードウェア構成が変更された場合に，R² 自体を容易に再構築することを可能としている．このように，R² は制御用ソフトウェアの移植性を向上させるために重要な役割を果たすと考えられる．

R² の設計目標は，実時間* システムに要求される高速応答性[11)~13)] の達成のみならず，容易に移植可能な（そして保守可能な）システムの実現にある．R² は，この応答性と移植性の2つの目標を達成するためのアーキテクチャを有している．本論文では，R² における種々の移植性向上策に関して述べる．R² のもう1つの目標である応答性に関しては，性能評価も含めて R²-86 核のアーキテクチャに関する論文で発表する．

以下，本論文では，最初に R² 開発の背景と設計目標について述べる．次に，全体のアーキテクチャの概要を設計目標の1つである移植性に焦点を当てて述べる．さらに，R² 自体の移植性向上策である機械依存部の記述方式と，R² を使用したアプリケーションプログラムの移植性向上策であるロボットプリミティブについて述べる．最後に，R² の開発環境について述べる．

2. 開発の背景

本章では，R² 開発の背景となった制御用ソフトウェア，特にロボットシステムのソフトウェアにおける一般的な要求と，これまでに我々が開発してきたロボットシステムにおける問題点についてまとめる．

2.1 ロボット制御用ソフトウェアにおける要求

一般に，産業用ロボットシステムは図1に示すように，ロボット本体，センサ，ロボット制御装置の3つ

† The Development of Real-Time Operating System R²-86 and Its Portability by EIJI OKUBO, TAKAO TSUDA, SYUZO KUSUDA, MASANORI KOBAYASHI (Department of Information Science, Faculty of Engineering, Kyoto University), KUNIHIKO SUGIMURA, KAZUTO SHIRAHAMA and YASUNOBU TOMODA (Mechanics and Electronics Division, Daihen Corporation).

†† 京都大学工学部情報工学科
††† (株)ダイヘン メカトロ事業部

* 「実時間」の範囲は，参考文献 1) に述べられているように広い範囲に渡っているが，R² では主にロボットシステムを対象としており，非同期に発生する事象に対する応答時間の限界として数十ミリ秒の範囲のシステムを考えている．

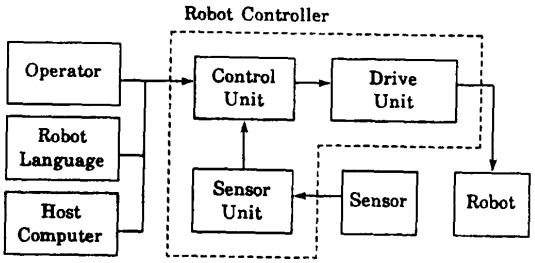

図 1 ロボットシステムの構成
Fig. 1 Configuration of robotic system.

の構成要素からなると考えられる（例えば文献 2）参照）．ロボット制御装置は，ロボットが使用される環境に対応して，視覚，触覚，動作等のための種々の処理ユニットが付加される．

ロボットシステム内部の制御方式に関しては，集中制御方式と分散制御方式の2方式が考えられる．どの方式を採用するかは，処理速度，ハードウェアコスト，保守性等を考慮して決定されている．集中制御方式のシステムでは，単一のプロセッサがすべての処理を行う．したがって，処理速度を向上させるためには，各処理単位に割当てるタスクの構成，特に時間的なタスクの生成と消滅のパターンが問題となる．しかし，そのスケジュールは極めて困難であり，経験的に行われているのが現状である．一方，分散制御方式のシステムにおいては，ロボット制御における各機能単位を単一のプロセッサに割付け，マスタプロセッサがそれらの機能単位に対応するスレーブプロセッサをスケジュールする形態をとる．この場合，システムの処理速度の観点からプロセッサ間で共用メモリを使用する，いわゆるマルチプロセッサ構成をとるものが多い．しかし，将来的には工場内はネットワーク化されることが予想され[14]，共用メモリを持たない真の意味の分散処理（すなわち，メッセージ指向の処理）が主流となり，より一層タスクの同期と通信方式の検討が必要になると考えられる．

以上のプロセッサ構成方式のほかに，プロセッサ自体にも種々のアーキテクチャが存在する．さらに，教示データや生産管理データの格納媒体である磁気バブルやハードディスク等と主記憶の間のメモリ階層の構成方式をも考慮する必要があり，多岐にわたるシステム構成に対処可能なソフトウェアが強く望まれていると考えられる．

2.2 従来のソフトウェアにおける問題点

我々は，R² の開発に際し，これまでに開発してきた制御用ソフトウェア，特に多関節型の溶接用ロボットのソフトウェアにおける問題点を検討した．その結果を，問題点とその対策に分けて**表1**に示す．表1に示すように，設計・開発・テストおよび保守の各段階で多くの問題点が存在する．また，これらの各段階に要した工数を比較すると，テストおよび保守にかかる工数が全工数の 50%を越えていることが分かった．これは，テストおよび保守段階においては，ロボットシステムが採用するプロセッサに強く依存していることに原因があると考えられる．すなわち，プロセッサ

表 1 従来の制御用ソフトウェアにおける問題点とその対策
Table 1 Problems and their solutions in the conventional control software.

	問題点	対策
設計	・要求仕様の記述があいまいである ・仕様分析が不十分である ・環境調査に時間がかかる ・設計範囲が広い ・ドキュメント量が多い ・思考が統一されていない ・ロボットの機能変更に柔軟に対応できない	・設計基準確立 ・ウォークスルーの強化 ・プログラムライブラリのツール化 ・モジュール化設計 ・プログラムの再利用 ・ロボット設計辞書の作成 ・基本機能と応用機能の分離
開発	・コーディングの量が多い ・アセンブル，コンパイル，リンクが遅い ・リンクパラメータやバージョンを間違う	・プログラムの再利用 ・高速ツールの採用 ・ライブラリおよびバージョン管理ツールの使用
テストおよび保守	・テスト準備に時間がかかる ・テスト項目が多い ・プログラムが複雑でデバッグに時間がかかる ・マルチタスクや割込みのテストができない ・ICE やデバッガを外付けしなければならない ・実機を使用しないとテストできない項目が多い	・テストツールの開発 ・テストツール，高級言語によるデバッグ ・モジュール化設計 ・シミュレータの開発 ・デバッグ機能の内蔵 ・シミュレータの開発

あるいはロボット本体が変更される度に新たに制御用ソフトウェアを開発しなければならず，既存のソフトウェアが有効に利用できていない点に原因がある．また，実環境でテストしなければならない項目があり，時間のかかる実機テストにどうしても頼らざるを得ないのが現状である．これは，ソフトウェアコスト，さらにシステム全体の開発コストや価格に大きな影響を及ぼす．このため，R² では，アプリケーションプログラムのみならず OS 自体の移植性の向上と，実機を使用する場面を極力少なくするためのデバッガおよびシミュレータを中心としたシステム開発支援機能を充実させることに目標を設定した．すなわち，表1に示す制御用ソフトウェアのライフサイクルにおける問題点のうち，テストおよび保守段階の問題点を解決することを主な目標としている．

3. 設計目標

前章の開発背景でも述べたように，我々は，実時間システムに要求される高速応答性の達成のみならず，容易に移植可能なシステムの実現を R² の目標として設定した．具体的には，ロボット制御に必要な，既存 OS では不十分な以下の点を解決することに目標を設定している．

（1） プロセッサ間の移植性が悪い．割込み処理，例外処理，さらにプロセッサ状態の制御等のプログラムの機械依存部を記述するに十分な機能を装備していない．さらに，バイトやビット単位のディスクリートな入出力がサポートされていない．

（2） ロボットとプログラム作成装置，あるいはロボットとホスト計算機との接続における通信制御のサポートがない．サポートされているとしても，各システムの入出力ドライバとして位置付けられている．

（3） 既存システムはシングルプロセッサ構成では使用可能であるが，マルチプロセッサ構成や分散型の構成の場合，システム全体のタスク構成，さらにそれらの通信と同期の方式をユーザ側で意識して管理しなければならない．

（4） 現在主流をなしているティーチングプレイバック方式のロボットは教示データを何らかの2次記憶に格納し，それを読み出して解釈実行するものであるから，このアクセス速度がシステム性能を決定する1つの重要な因子となっている．しかし，既存システムでは，プログラム開発のためのインタフェースとしてファイル管理機能がサポートされているが，実時間の入出力に関しては何らの対処も施されていないのが現状である．

（5） ターゲットプロセッサが変更されると，そのための開発システムも変更しなければならない．すなわち，開発装置自体がプロセッサに依存している．

R² では，（1）のプロセッサ間の移植性の問題を解決するために，ユーザ固有の割込み処理や入出力ドライバをC言語で記述するための機能をサポートしている．さらに，R² 自体の移植性を向上させるために，R² を構成する各モジュールの機械依存部を独立したファイルとして構成することによって，他プロセッサへの R² の移行時に，それらが容易に識別可能となるようにしている．（2）および（3）に関しては，R² 標準でマルチプロセッサ構成のための通信管理およびタスク管理を実現している．アプリケーションプログラムでは，タスクの同期および通信においてプロセッサ構成を意識したくない．このために R² では，相手タスクの識別子を指定するだけで同期および通信が行える機能を提供している．（4）に関しては，ファイル管理を標準でサポートし，ディレクトリを主記憶に常駐させることによって高速アクセスを実現している．（5）に関しては，2.2 節で述べたように，デバッガおよびシミュレータを中心としたシステム開発支援機能を充実させている．

4. R²-86 のソフトウェア構成

R²-86 は，マイクロプロセッサ 8086/8088/80186 上で動作可能な実時間 OS である．ただし，ハードウェアとしてインターバルタイマおよび割込みコントローラの装備を前提としている．R²-86 は，RAM ベースおよび ROM ベースの両方の型のシステムとして生成することが可能である．R²-86 は，主にロボット制御を対象に考えており，R²-86 標準ではロボットのティーチングデータの管理やロボットプリミティブ等のロボットに関する処理の専用機能を有している．しかし，システムの基本部分である R²-86 核は実時間制御が行われる 8086 ベースの任意のシステムで使用可能である．

R²-86 システムは，以下のソフトウェアから構成される（**図2**参照）．

（1） R²-86 核
（2） 入出力ドライバ
（3） ファイル管理
（4） ロボットプリミティブ

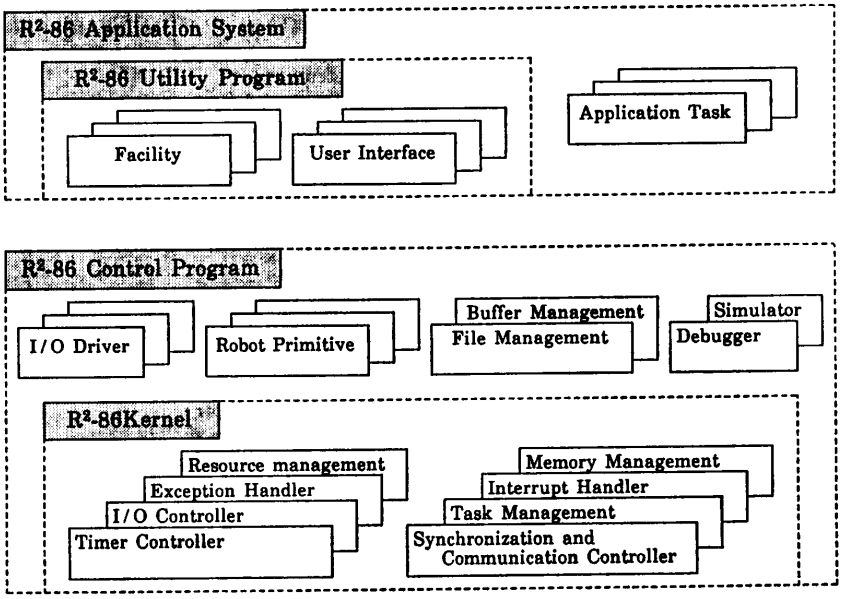

図 2　R²-86 のソフトウェア構成
Fig. 2　Software configuration of R²-86.

（5）　ファシリティ(アプリケーションプログラム)
（6）　ユーザインタフェース
（7）　デバッガおよびシミュレータ

特に，本構成法と従来の構成法の相違点は，ロボットプリミティブを設定した点にある．以下，各構成要素について説明する．

（1）　R²-86 核

R² では，応答性と移植性の２つの目標を設定している．そのため，R²-86 核では，高速応答性に関してタスクディスパッチング処理と割込み処理の高速化を実現している．タスクディスパッチング方式としては，事象駆動 (event driven) 方式を採用し，高速処理を必要とするタスクを優先度の高いタスクとすることによって応答性に優れたシステムを容易に構成することを可能としている．また，割込みに関しては，M状態およびE状態の２つの割込み処理の状態を設けている．M状態においては，割込みをネストさせることによって，当該割込みよりも優先度の高い割込みが即時処理される．一方，E状態では，割込みタスクとして割込み処理が行われ，この間すべての割込みが許可されるので割込み禁止時間を短縮することが可能である．M状態およびE状態の割込み処理の記述例をそれぞれ図3(d) および (e) に示す（図３全体は，次に述べる入出力ドライバの記述例となっている）．移植性に関しては，機械依存部を除いて R²-86 核自体もC言語で記述されている．また，機械依存部である割込みハンドラに関しては，図３(c) に示すようにレジスタの追避・回復，スタックの初期設定等の割込み処理の環境設定のためのC言語インタフェースを持つライブラリ関数を提供している（図３(c) の Xsetenv_m() および Xexit_m()）．

（2）　入出力ドライバ

ハードウェアからの独立性を向上させるため，すなわちハードウェア変更に伴うアプリケーションプログラムの変更を緩和するため，各種入出力をすべて統一して扱えるインタフェースを提供している．具体的には，バイトやビット単位の入出力，A/D および D/A，CRT，磁気バブル，さらに通信のための種々の入出力を，SIO（標準入出力），DIO（ディスクリート入出力），BIO（ビット入出力），WIO（ワード入出力）の４つの SVC によって同様の形式で記述することが可能である．さらに，割込みハンドラと同様，機械依存部であるユーザ固有のドライバをC言語で記述するためのライブラリを提供している．各ドライバは，初期化ルーチン，SVC エントリルーチン，割込みハンドラ，M状態割込み処理ルーチン，E状態割込み処理ルーチン，タイムアウトルーチンの６つのルーチンから構成され，これらのルーチンはすべてC言語で記述可能である．図３にドライバの記述例を示す（ただし，タイムアウトルーチンは省略している）．図３の各ルーチンで使用されている大文字で始まる関数は，ドライバや割込み処理の記述のために R² で提供して

```
init_drv ( ucb_ptr )
struct ucb    *ucb_ptr;
{
    bo();    /* I/O port initialize */
    . . .
    bo();
    . . .
    ucb_ptr → sts = NOT_BUSY;
}
```

(a) 初期化ルーチン

(a) Initialize routine.

```
svc_drv ( irb_ptr, func_code )
struct irb    *irb_ptr;
int           func_code;
{
struct ucb    *ucb_ptr;
/* get UCB pointer */
    ucb_ptr = irb_ptr → ucb_ptr;
/* decode function code */
    switch ( func_code ) {
    case READ:
        . . .
        break;
    case WRITE:
        . . .
        break;
    default:
        Dio_end ( ucb_ptr, FUNC_ERR );
        Dnext_svc ( ucb_ptr );
        break;
    }
}
```

(b) SVCエントリルーチン

(b) SVC entry routine.

```
Xentry_drv ()
{
    Xsetenv_m();
    m_int_drv ( dev_no, unit_no );
    Xexit_m();
}
```

(c) 割込みハンドラ

(c) Interrupt handler.

```
m_int_drv ( dev_no, unit_no )
int  dev_no, unit_no ;
{
struct ucb    *ucb_ptr;
/* get UCB pointer */
    ucb_ptr = Dgetucbptr ( dev_no,
                           unit_no );
    . . .
/* create interrupt task */
    Xfork ( e_int_drv, param );
}
```

(d) M状態割込み処理ルーチン

(d) M-state interrupt routine.

```
e_int_drv ( param )
char  *param;
{
    . . .
    Dnext_func ( ucb_ptr );
}
```

(e) E状態割込み処理ルーチン

(e) E-state interrupt routine.

図 3 入出力ドライバの記述例

Fig. 3 Example description of I/O driver.

いるライブラリ関数である.

(3) ファイル管理

R² では, 実時間 OS では十分にサポートされていないファイル管理の機能を標準でサポートしている. ファイル管理では, ディレクトリを主記憶に常駐させることによって, 2次記憶の高速アクセスを実現している. さらに, R² のファイル管理では, プログラム開発用の小型 OS である MS-DOS のファイル構造(階層的ディレクトリ構造)を管理する機能をサポートしている. さらに, バッファ管理機能およびティーチングデータの管理機能も標準でサポートしている. ティーチングデータは, ロボットの動作を記述するものであり, 一般にフロッピーディスクや磁気バブル等の外部記憶に格納されている. R²-86 では, このティーチングデータに対するランダムアクセスとインデックスによるアクセスをサポートしている.

(4) ロボットプリミティブ

R² では, アプリケーションプログラムの可読性を向上させ, さらにソフトウェアの再利用を可能とするために, ロボットプリミティブと呼ばれるC言語の関数を基本としたシステム構成法を採用している.

ロボットプリミティブは, これまでに我々が開発してきたロボット制御用プログラムから共通的でかつ普遍的な機能を抽出し, ライブラリとしてまとめたものである (5.2 節参照). これには, ロボットの動作プリミティブ, 座標変換用プリミティブ, ティーチング用プリミティブ等がある.

(5) ファシリティ

R² では, ロボット用アプリケーションプログラム(データ編集, 異常処理, ティーチング, ブロック運転, 自動運転等) をファシリティと呼び, これらのプログラムをロボットプリミティブの系列を使用して構

成している．これにより，ファシリティレベルの移植性が大幅に向上すると考えられる．

（6） ユーザインタフェース

ロボット言語用コンパイラを初めとしたオフラインプログラミング機能，ティーチングボックスによる教示，実行状況の表示および稼動管理情報の編集・出力といった生産管理等のプログラムがある．

（7） デバッガおよびシミュレータ

R²-86 では，システムのモニタリングおよびデバッグ機能を内蔵している．さらに，ターゲットプロセッサの動作をシミュレートするためのソフトウェアシミュレータを R² ファミリとしてサポートしている．このソフトウェアシミュレータは，VAX-11/750 (OS は UNIX 4.2 bsd) 上で動作し，マルチタスク環境をシミュレートすることが可能である．これらの機能により，アプリケーションプログラムを作成するための専用のプログラム開発装置が不要になると考えられる．

5. R² における移植性向上策

R² 上のアプリケーションプログラムは，C 言語で記述することを想定しているので，各種 SVC やライブラリのインタフェースをC言語の仕様に合わせている．また，機械依存部のアセンブリ言語記述を除いてシステム自体もC言語で記述されており，言語レベルでは移植性の良いシステムであると言えよう．本章では，R² における移植性の向上策として採用したプログラムの機械依存部の記述方式とロボットプリミティブについて述べる．

5.1 プログラムの機械依存部の記述方式

一般に，プログラムの移植性を向上させるための手法として以下の 4 つが考えられる．

（1） 高級言語による記述

（2） 実行環境に依存する部分と依存しない部分のファイルレベルでの分離

（3） システム記述言語におけるコンパイラ制御行の使用

（4） システムの構造記述（移植性の高いアーキテクチャ）の実現

(1)の方法は，最近ではごく普通に行われている．移植性に優れた OS という場合，高級言語により記述されたものを指すことが多い．UNIX[15] は，この種の OS として代表的なものである．(2)の方法は，プログラムを実行環境に依存するモジュール集合と依存しないモジュール集合に分割し，かつそれらを格納するファイルも別のものとする方法である[16]．すなわち，プログラムを，各実行環境に対応して変更され得る要素（環境変数と呼ぶ）を含むものと含まないものに分割することである．例えば，CP/M では，基本的な入出力ルーチン群を BIOS として構成し，他のモジュールとファイルを分離することによって移植性を向上させている（この場合の環境変数は，入出力機器となる）．(3) の方法は，移植可能な環境に対応して必要な処理をすべて記述する方法である．C言語における #ifdef 文等がこれに該当する．UNIX は，この手法を使用して機械に依存したモジュールを記述している．(4) の方法は，タスクの構造等に関してアーキテクチャレベルから移植性の高い構造を実現しようとするものである[17)~19)]．

以上 4 つの方法が考えられるが，R² では 2 番目の方法を採用している．R² では，実行環境に“依存する”および“独立である”という言葉の意味を以下のように定義している．

依存：環境が変更された場合，当該モジュールがある環境変数に関して無効となるとき，その環境変数に関して“依存する”と呼ぶ．

独立：環境が変更されても，当該モジュールが依然と

File1 "def.h"

```
#define     SLENGTH     1024
```

File2

```
#include    "def.h"
char        buf[SLENGTH]
main()
{
int     ret;
        ret = dread(3, buf);
        printf("/n returns: %s", ret);
}
```

File3

```
dread(sec, ptr)
int         sec;
char        *ptr;
{
struct regval {int ax, bx, cx, dx, si, di, ds, es;} allreg, retreg;
        sysint(0x1b, &allreg, &retreg);
        return(retreg.ax);
}
```

図 4 機械依存部の記述例
Fig. 4 Example description of machine-dependent part.

してある環境変数に関して有効である場合，その環境変数に関して“独立である”と呼ぶ．

環境変数としては，以下のものが考えられる．

（1） プロセッサ

（2） 入出力機器

（3） OS

（4） システム記述言語

これらの環境変数に基づいてファイル分割を行うことによって，プログラムの移植性が向上する．すなわち，プログラムを移植する場合，もとの環境と新しい環境において可変である環境変数を調べ，それに関して依存するファイルのみを変更し再びコンパイルすることで移植が完了する．この方法の主眼は，環境に依存するモジュールを特定のファイルに局所化することにある．例として，**図4**のディスクから指定されたセクタの内容を読み出すためのアプリケーションプログラムの記述を考える．図4の例で，ファイル1は明らかに実行環境であるディスクコントローラに依存したファイルである（SLENGTH の値がディスクコントローラに依存している）．ファイル3は，レジスタやシステムコールを記述しているため，プロセッサおよびOSに依存したファイルである．さらに，Optimizing-C86 コンパイラにおける関数 sysint() を使用しているので，記述言語にも依存している．しかし，ファイル2は，環境にまったく依存しないファイルであると言える．#include によりファイル1をテキスト上に取り込み，さらにファイル3の関数 dread() を呼び出しているが，#include や dread() といった記述自体は環境に依存しないからである．したがって，このプログラムをある環境から他の環境へ移植する場合，ディスクコントローラ（入出力機器）のみが異なるならば，ファイル1のみを変更し再コンパイルすればよい．また，記述言語が異なっている場合はファイル3を変更すればよい．

この方法により移植性を向上させるためには，単に作成したモジュール集合をその依存性に基づいてファイル分割するだけでは不十分である．それは，各モジュールが複数の環境変数に関して依存する可能性があることによる．しかし，モジュールを作成する時点で依存性を考慮すれば，1つのモジュールが依存する環境変数の種類を大幅に減少させることが可能となる．さらに問題となるのは，どのようなものを可変の環境変数として扱うかという点である．換言すれば，プログラムの移植の条件としてどのような制限事項を設定するかということである．もちろん，より多くの環境変数を可変にすることによって移植可能な環境が拡大される．しかし，極端な依存性に基づくモジュール分割およびファイル分割を行うと，システムの性能の低下の原因になりかねない．モジュール設計を行う場合，以上の点に注意することが必要である．

5.2 ロボットプリミティブ

従来のロボット制御用プログラムは開発時における要求に応じて，また各機械に対してその都度設計を行っているのが現状である．一方，ロボット言語の標準化が進められており，これらの制御用ソフトウェアに対して将来の標準化された言語への対応も求められている[3]．このような背景の下で，我々は，ソフトウェア再利用の観点から，これまで開発してきたロボット制御用プログラムからロボット固有の基本的機能を抽出し，ロボットプリミティブとしてまとめた．このロボットプリミティブにより，アプリケーションプログラムにおける機能の冗長性の除去（機能の明確化），標準化されたロボット言語への対応が容易となる．本節では，これらのロボットプリミティブの構成法について述べる．

（1） ロボットプリミティブの構成

ロボットプリミティブは，処理プリミティブとデータプリミティブから構成される．処理プリミティブとは，直線や円弧等の補間機能や手動運転等のロボット本体の基本動作機能のほか，ティーチングデータの記憶，追加，変更，削除といったティーチング機能等のロボット固有の処理をパッケージ化したものである．データプリミティブは，処理プリミティブで操作されるロボット固有のデータである．これらのプリミティブは，すべてC言語で記述されている．**表2**に R^2 プロジェクトの一環としてこれまでに作成した各機能ごとのプリミティブの個数を示す．これらのプリミティブは，マニピュレータ型のロボットの動作とティーチ

表2 機能別のロボットプリミティブの個数
Table 2 Number of robot primitives classified by functions.

機能	個数
電源投入	14
原点復帰	7
手動運転	13
ブロック運転	49
自動運転	14
ティーチング	20
例外処理	4

```
VOID    power_on()
{
        if ( p_chk_ram() != 0 ) {
                if ( p_dsp_abnml_ram() != 0 ) p_crterr();
                if ( p_input_downcont() != 0 ) p_sys_down();
                if ( p_erase_abnml_ram() != 0 ) p_crterr();
        }
        if ( p_chk_comm() != 0 ) {
                if ( p_dsp_abnml_comm() != 0 ) p_crterr();
                if ( p_input_downcont() != 0 ) p_sys_down();
                if ( p_erase_abnml_comm() != 0 ) p_crterr();
        }
        p_init_data();
        p_read_sys_data();
        if ( p_dsp_hret_guide() != 0 ) p_crterr();
}
```

(a) 電源投入処理の記述例
(a) Example description of power-on processings.

```
VOID    home_ret()
{
        setexcp( CONDITION, moveerr, NOCHANGE );
        time_clock = 30;
        time_unit = 2;
        if ( p_erase_hret_guide() != 0 ) p_crterr();
        if ( p_dsp_hret() != 0 ) p_crterr();
        if ( p_start_hret() != 0 ) raise( CONDITION );
        pause( time_clock, time_unit );
        p_set_hret();
        if ( p_erase_hret() != 0 ) p_crterr();
}
static VOID     moveerr()
{
        p_sys_down();
}
```

(b) 原点復帰処理の記述例
(b) Example description of home-return processings.

図 5 ロボットプリミティブを使用したファシリティの記述例
Fig. 5 Example descriptions of facilities using robot primitives.

ングに関係する基本的な物のみであり，これからさらに充実させていかなければならないと考えている．

ロボットプリミティブを使用したアプリケーションプログラムの記述例を図5に示す．これは，電源投入および原点復帰を記述した例である（ただし，宣言部は省略している）．図中，“p_”で始まるものがプリミティブである．すなわち，各プリミティブはC言語の関数として構成されている．したがって，その呼び出しオーバヘッドがシステム性能に影響すると考えられる．しかし，C言語における関数呼び出しのオーバヘッドを測定した結果(使用言語；OPTIMIZING-C86，測定機器：PC 9801 M2)，1つのプログラムで呼び出す関数の数を 20 個以下とすると，そのオーバヘッドは636マイクロ秒以下（引数3個の場合）となり実用上問題ないことが分かっている．

（2） ロボットプリミティブの作成法

我々は，ロボットプリミティブを作成するにあたって以下の基本方針をとった．

（a） 全体のプログラムの流れで，分岐の多い制御シーケンスは，アプリケーションプログラム側で記述する．すなわち，1つのロボットプリミティブの中で多ケースを扱うのではなく，各ケースごとにプリミティブを作成する．

（b） 自動運転，ブロック運転，ティーチング等の異なる機能に属すプリミティブが同一のデータプリミティブを参照しないようにする．すなわち，同一の機能に属すプリミティブと，それらが操作するデータを明確に対応付けるようにする．

（c） ハードウェアの機能分割とロボットプリミティブの機能分割を1対1に対応させる．すなわち，機能分散のシステムにおいて，1つのプリミティブが複数のシステムにまたがらないようにする．

以上の方針を設定した理由は，それぞれ以下のことによる．

（a） 1つのプリミティブ内で多ケースを扱うことによってコード量が増えるのを避ける．さらに，他のプリミティブを変更することなしに機能追加を行えるようにする．

（b） データの変更が複数のロボットプリミティブに影響するのを避ける．

（c） ハードウェアの変更に対しロボットプリミティブの対応を容易にする．

6. 移植性の評価

R² は OS であるから，R² 自体の移植性を考えるとき環境変数として OS は含まない．表3に R²-86 核および入出力ドライバを構成するモジュールのオブジェクトサイズ，ステップ数，環境変数等の統計情報の一覧を示す．表3におけるアセンブリ言語記述のモジュールは，命令セットに依存するという意味で，プロセッサを環境変数として記載している．表3より，R²-86 核の環境変数としてはプロセッサがあり，約1割が環境に依存することが分かる．また，ドライバの環境変数としては入出力機器があり，そのすべてが環境に依存している．ただし，これらのドライバはほとんどがC言語で記述されているので，従来の OS に

表 3 R²-86 核および入出力ドライバのファイル統計一覧
Table 3 List of file statistics in R²-86 kernel and I/O drivers.

モジュール	ファイル数	関数の個数	記述言語	オブジェクトサイズ(バイト)	ステップ数	環境変数
タスク管理	2	35(1)	C	5011	562(15)	プロセッサ
メモリ管理	2	28	C	8751	782	
同 期	2	10	C	785	134	
通 信	2	30	C	4771	664	
CPU 間通信	2	31	C	5614	686	
資源管理	2	5	C	1096	116	
割込み制御	2	4(1)	C	200	164(9)	プロセッサ
	3	24(24)	ASM	1358	672(672)	プロセッサ
入出力制御	2	17	C	4638	460	
タイマ管理	2	14	C	2021	328	
例外処理	2	38(1)	C	4712	578(12)	プロセッサ
	2	4(4)	ASM	71	56(56)	プロセッサ
小 計	25	240(31)		39128	5202(764)	
ドライバ	5	56(56)	C	25744	2107(2107)	入出力機器
	2	10(10)	ASM	146	110(110)	入出力機器
合 計	32	306(97)		65018	7419(2981)	

注1) ステップ数は，注釈行，空白行，include 文，define 文，extern 文を除く．
注2) 各項目の括弧内の値は，環境変数に依存するものの値である．
注3) ドライバは，FDC, PIT, キーボード，CRT，通信用の計5つである．

表 4 R²-86 核のヘッダファイルの環境変数
Table 4 Environment variables in header files of R²-86 kernel.

ファイル名	環境変数	参照ファイル数	内 容
def.h	記述言語	25	定数および型の定義
excond.h	プロセッサ	3	例外条件の定義
exfunc.h	記述言語	2	例外ハンドラの外部参照の定義
machine.h	プロセッサ	8	プロセッサ固有の定数の定義
rsa.h	プロセッサ	1	レジスタ退避領域の定義
tcb.h	プロセッサ	24	TCB の定義

おけるアセンブリ言語記述のドライバよりは移植性が向上していると言えよう．

表4には，R²-86 核のヘッダファイルのうち，環境に依存するものを記載している．表4において，環境変数がシステム記述言語となっているのは，パラメータや関数の戻り値の渡し方等が，使用するC言語の処理系によって各々異なっているからである．表4の各ファイルは，R²-86 核の複数のモジュールに include されているものである．したがって，R² において機械依存部のファイル分割の方式を採用しなかった場合は，これらが include されている各モジュールに展開されることとなる．したがって，機械依存部のモジュールが大幅に増えることになる．アプリケーションプログラムに関しては，実際の場面に R² を適用していないので，表3および表4のようなデータは採取できないが，機械依存部である割込みハンドラ，デバイスドライバ，例外処理のC言語記述が可能なので，従来の OS よりも移植性は向上していると考えられる．

7. R² の開発環境

R² は，VAX-11/750 (OS は UNIX 4.2 bsd) 上で開発されている．図6に R² システムのディレクトリ構造を示す．ディレクトリは，大きく OS とプリミティブの2つに分かれている．さらに，前章で述べた機械依存部のファイルは，このディレクトリ構造において物理リンク (physical link) で互いに結ばれているため，容易に識別可能となっている．

VAX 上では，各モジュールのコンパイル，モジュール間インタフェースの整合性チェック，各種ツー

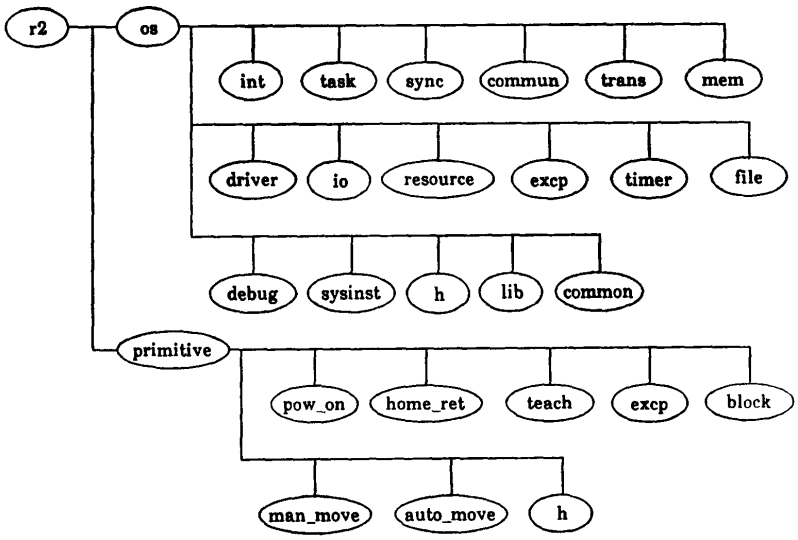

図 6 R²-86 システムのディレクトリ構造
Fig. 6 Directory structure of R²-86 system.

ルを利用したテストを行っている．特に，複数モジュールの結合テストに関しては，R² 専用の環境を構築するための小さなシミュレータを使用している．さらに，ロボットシステムの実行システムとしては，三菱電機（株）製ムーブマスターⅡを PC 9801 に接続したシステムを使用している．R² 自体のシステム生成は，VAX と PC 9801 間でソースレベルのファイル転送を行い，PC 9801 上の MS-DOS を使用して行っている．

8. おわりに

以上，本論文では応答性と移植性を目標とした実時間制御用の OS である R²-86 の移植性に焦点を当てて述べてきた．現在（1986 年 12 月），その OS 核とロボットプリミティブを中心としたシステムの基本部分の第 1 版の開発を終了した段階である．今回実現された R²-86 の大部分はC言語により記述されており，当初の目的である移植性の高いシステムの実現は十分に達成されたと考えている．R²-86 核は，要求仕様の検討および設計段階も含めて 4 人・年で開発された．今後は，R²-86 を使用した各種アプリケーションプログラムを開発し，評価を行っていかなければならないと考えている．このため，現在，2 台の PC 9801 とロボットコントローラおよびロボット本体各 1 台のプロトタイプシステムを構築中である．

謝辞　本システムの要求仕様の検討から開発に至るまで貴重な御意見を頂いた村上悦三氏を初めとする株式会社ダイヘンのメカトロ事業部の各位に感謝の意を表する次第である．

参考文献

1) Glass, R. L.: *Real-Time Software*, Prentice-Hall, New Jersey (1983).
2) 情報処理：大特集：ファクトリオートメーション，Vol. 25, No. 4 (1984).
3) 日本機械工業連合会，日本産業用ロボット工業会：昭和 58 年度産業用ロボット標準化推進事業報告（ロボット標準言語の作成）(1984. 5).
4) 高田昌之，石川　浩：ロボット用オペレーティングシステムに望まれるもの，日本ロボット学会誌，Vol. 3, No. 5, pp. 26-31 (1985).
5) 三浦宏文，下山　勲，木村　浩：リアルタイム OS を用いたロボットの制御，日本ロボット学会誌，Vol. 3, No. 5, pp. 32-40 (1985. 10).
6) 大久保英嗣，津田孝夫，楠田修三，小林正典，杉村邦彦，白濱和人，友田和伸：実時間オペレーティングシステム R² の設計目標と全体構成，第 33 回情報処理学会全国大会講演論文集，3V-6 (1981. 10).
7) 楠田修三，大久保英嗣，津田孝夫，小林正典，杉村邦彦，白濱和人，友田和伸：実時間オペレーティングシステム R² のタスク管理方式，第 33 回情報処理学会全国大会講演論文集，3V-7 (1981. 10).
8) 杉村邦彦，白濱和人，友田和伸，大久保英嗣，津田孝夫，楠田修三，小林正典：実時間オペレーティングシステム R² の通信管理方式，第 33 回情報処理学会全国大会講演論文集，3V-8 (1981. 10).

9) 白濱和人，杉村邦彦，友田和伸，大久保英嗣，津田孝夫，楠田修三，小林正典：実時間オペレーティングシステム R² の入出力制御方式，第 33 回情報処理学会全国大会講演論文集，3V-9 (1981. 10).
10) 友田和伸，杉村邦彦，白濱和人，大久保英嗣，津田孝夫，楠田修三，小林正典：実時間オペレーティングシステム R² のロボットプリミティブとプロトタイプシステム，第 33 回情報処理学会全国大会講演論文集，3V-10 (1981. 10).
11) 日経エレクトロニクス：機種対応に顔を揃えた 16 ビット・マイクロコンピュータ用 OS—充実するリアルタイム制御用 OS, pp. 122-136 (1981. 10).
12) 加藤肇彦，筒井茂義，茶木英明：マイクロコンピュータ用オペレーティングシステムの現状と動向，情報処理，Vol. 25, No. 3, pp. 232-243 (1984).
13) 坂村 健：リアルタイムオペレーティングシステム—ITRON，情報処理学会オペレーティングシステム研究会，24-10, pp. 59-64 (1984. 9).
14) MAP JAPAN MEETING '85: What Is MAP and Its Impact (1985. 8).
15) Ritchie, D. M. and Thompson, K.: UNIX Time-Sharing System, *CACM*, Vol. 17, No. 7, pp. 365-375 (1974).
16) Cheriton, D. R., Malcolm, M. A., Melen, L. S. and Sager, G. R.: Thoth, a Portable Real-Time Operating System, *CACM*, Vol. 22, No. 2, pp. 105-115 (1979).
17) Zwaenepoel, W. and Lantz, K. A.: Perseus: Retrospective on a Portable Operating System, Technical Report, No. STAN-CS-83-945 (1983), Computer Systems Laboratory Departments of Electrical Engineering and Computer Science, Stanford University.
18) Crowley, C.: The Design and Implementation of a New UNIX Kernel, *NCC*, pp. 265- (1981).
19) Tsutsui, S., Yokohata, S., Hatada, M., Watanabe, T., Katoh, H. and Yabe, E.: Universally Interfaceable Modular Operating System for 68000 Microcomputers, *COMPCON FALL*, pp. 453-460 (1983).

(昭和 61 年 7 月 7 日受付)
(昭和 62 年 4 月 15 日採録)

大久保英嗣（正会員）

昭和 26 年生．昭和 49 年北海道大学理学部数学科卒業．昭和 52 年同大学工学部情報工学科大学院修士課程修了．同年(株)日立製作所ソフトウェア工場に入所．主として FORTRAN コンパイラの開発に従事．昭和 54 年より京都大学工学部情報工学科助手．昭和 60 年同講師，昭和 62 年同助教授，現在に至る．工学博士．オペレーティングシステム，データベースシステム等の研究に従事．日本ソフトウェア科学会，日本ロボット学会各会員．

津田 孝夫（正会員）

1932 年生．1957 年京都大学工学部電気工学科卒業．現職は京都大学工学部情報工学科教授．工学博士．現在の主要研究テーマは，メモリ階層間データ転送量の下界とそれによるアルゴリズムの最適化，ベクトル計算機のための自動ベクトル化の自動並列化，実時間オペレーティングシステムなど専用 OS の構成と実現法など．

楠田 修三（正会員）

昭和 37 年生．昭和 60 年京都大学工学部情報工学科卒業．昭和 62 年同大学院修士課程修了．同年日本アイ・ビー・エム(株)入社．オペレーティングシステム，コンピュータネットワーク，プログラミング言語等に興味を持つ．

小林 正典（学生会員）

昭和 38 年生．昭和 61 年京都大学工学部情報工学科卒業．同年同大学院修士課程入学．リアルタイムオペレーティングシステム，分散処理等に興味を持つ．

杉村 邦彦（正会員）

昭和 27 年生．昭和 51 年大阪大学基礎工学部制御工学科卒業．昭和 53 年同大学院修士課程修了．同年(株)ダイヘン入社．メカトロ事業部技術部に所属．産業用ロボットの制御用ソフトウェアの開発に従事．実時間システムのオペレーティングシステムおよびソフトウェア開発環境等に興味を持つ．

白濱　和人（正会員）

昭和34年生．昭和58年鹿児島大学工学部電気工学科卒業．同年(株)ダイヘン入社．メカトロ事業部技術部に所属．産業用ロボットの制御用ソフトウェアの開発に従事．プログラミング言語とその処理系，プログラミングツール等に興味を持つ．

友田　和伸（正会員）

昭和35年生．昭和59年愛媛大学工学部電気工学科卒業．同年(株)ダイヘン入社．メカトロ事業部技術部に所属．産業用ロボットの制御用ソフトウェアの開発に従事．実時間制御ソフトウェアに興味を持つ．