ソフトウェア工学 84－7
（1992. 3. 3）

組込みシステムにおける大規模ソフトウェア開発の一支援方式

大塚 壮一*，神田 正巳*，中本 幸一**

* 日本電気技術情報システム開発 (株)
** 日本電気 (株)C&C 共通ソフトウェア開発本部

従来，通信処理システムなど組込みシステム向けの大規模ソフトウェアは，その構造が複雑であり，修正時にプログラムの再リンク・再配置が必要となるためその変更が容易でなく，デバッグ時の作業効率も妨げていた．筆者らは，この問題を解決し，組込みシステムの一つである通信処理システム向け OS CTRON の拡張 OS，アプリケーションプログラムの開発を支援するために，ターゲットソフトウェアをモジュールという構成単位でモデル化し，このモデルによりソフトウェアの構成を記述する言語を導入した．更にモジュール単位でのプログラムのリンクを可能とし，デバッグ時にもこの単位でプログラムの一部の追加・修正を可能にするサポートシステムを開発した．

A Method to Support to Develop Very Large Software in an Embedded System

Souichi Ohtsuka*, Masami Kanda*, Yukikazu Nakamoto**

*NEC Scientific Information System Development, Ltd.
** C&C Common Software Development Laboratory, NEC Corporation

The complicated structure of large software in an embedded system like a communication system leads to being difficult to modify it in order to need to relink and relocate programs and the difficulty prevents efficient debugging. The authors solve these problems and model target programs as a *module* and define a language to specify configuration of the programs in terms of a module in order to support development of extended OS and application programs of OS, CTRON, for communication systems, one of the embedded system. Moreover they develop a support system which allows a user to link the program by a module and replace parts of the program and append the parts into the programs.

1 はじめに

組込みシステムに対するサービスの要求は複雑化，多様化し，こうしたサービスを実現するために，組込みシステムのソフトウェアは大規模化，複雑化する傾向にある．

一般に，情報処理システムのソフトウェアと異なる組込みシステムのソフトウェアの構成上の特徴として，以下の点が考えられる．

プログラムの再配置: プログラムがメモリに常駐されるため，システムを構成する全プログラムのシンボルにユニークなアドレスをロケータで割り当てる必要がある．このため，プログラムを修正，変更した場合，再配置しなければならない．このような事態を避けるため，プログラムを配置する場合に，プログラム間にギャップを空けて，プログラムのサイズ変更が生じても，その再配置が起こらないようにする手法がとられている (例えば，[1]).

プログラムの再リンク: 制御対象物の実時間制御や組込みシステムのハードウェアの制御等，複雑で高速な機能を実現するために，プログラム同士が，シンボルで互いに直接参照するなどシンボルの参照関係が複雑である．それゆえ，プログラムを修正，変更した場合に，プログラム間を再リンクする等の必要がある．

このように，組込みシステムでは，プログラムを変更した場合に，再リンクや再配置が生じるため，ロードモジュールを生成するのに多くの時間がかかる．更に，変更された部分に対応するロードモジュールのみを作成することが難しい場合は，システム全体のロードモジュールを作成してダウンロードすることになり，デバッグ時の，ソースファイル修正 → コンパイル → ダウンロード → デバッグのサイクルが長くなり，デバッグ効率を妨げる結果となる．

また，組込みシステムに対する要求に柔軟に対応するために，ソフトウェアの構成変更が容易にできることが要請されている．しかし，上記の2つの理由から，組込み型システムのソフトウェアの修正，構成変更が難しいとされている．

更に仮想空間管理機構を有する32ビットマイクロプロセッサを用いたシステムでは，ソフトウェアに対する要求は特に複雑化，多様化し，これを利用した交換機，通信処理システム等の通信処理を行なう組込みシステムでは，そのソフトウェア規模が大きく，前述した問題はさらに著しくなる．

筆者らは，通信処理用オペレーティングシステム CTRON カーネルを 32 ビット CISC プロセッサ V70，RISC プロセッサ VR-3000 用に開発した [3]．更に，これらのカーネルの上で動作する拡張 OS やアプリケーションプログラムの構築を容易にするツールを開発した．上述した問題を解決するために，筆者らは CTRON サポートシステムの設計目標を以下のように掲げた．

1. 組込みシステムのソフトウェアの構成，変更が容易であること．例えば，組込みシステムを構成するソフトウェアの一部が変更された場合でも，そのソフトウェア全体を再リンクする必要がないことなどがあげられる．

2. ロードモジュールの部分的な追加，入れ換え等ができること．これは，特にデバッグ時に修正，変更したロードモジュールの一部をロードすることにより，デバッグ時のソースファイル修正 → コンパイル → ダウンロード → デバッグのサイクルを短縮することが可能となる．

筆者らは，この設計目標に基づいて，まず組込みシステムソフトウェアのモデル化を行なった．次にこのモデルに基づき，組込みシステムソフトウェアの構成を記述する言語を定義し，この記述に基づき組込みソフトウェアを構築するサポートシステムを開発した．本稿では，VR-3000 用 CTRON サポートシステムについて述べる．

2 ターゲットシステムのモデル化

先に挙げた設計目標をもとに，ターゲットとなるシステムのモデル化をおこなった．

システム全体の構成 ターゲットシステムのソフトウェアは，カーネルと複数のモジュールから構成される (図 1).

カーネル リアルタイム OS のカーネルである．複雑化，大規模化する組み込みシステムに適

した仮想記憶管理機能を持つ．また，通常のタスク等の資源管理機能の他に，モジュールの管理機能を持つ．

CTRONでは，基本OSのカーネルに相当する[4]．

モジュール システムの機能を分担し独立性が高く，プログラムの変更や追加，ロケートの単位であり再配置可能である．

モジュールは，機能的にまとまったタスクやライブラリ関数の集まりとして実現される．そして，その機能は，関数呼び出し形式のインタフェースを介して利用される．

例えば，CTRONの拡張OSは，このようなモジュールの集まりとして定義することも可能である．

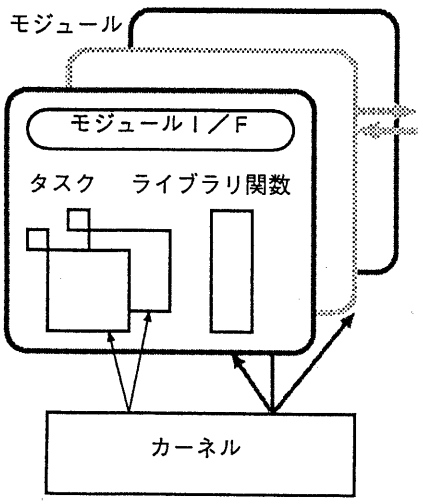

図 1: システム構成

3　システム構成定義記述言語

上記のモデルに従って実際のシステムを開発するためには，何をモジュールとするか，そのインタフェースは何かなど，モデルに合ったシステム構成の記述が必要である．

このシステム構成の定義(SG定義)は，モジュールの処理プログラムとは別に，専用のSG言語で記述する．定義内容は，物理空間の構成や，最大タスク数，モジュールの定義など，システムに依存して変化するデータ類であり，次の3種類の定義がある．

3.1　システム定義

物理空間や仮想空間の構成，各種ロードモジュールのロードアドレスなどを定義する(図2)．主な定義を以下に示す．

```
physmem { /* 物理空間構成の定義 */
    mem1 = 0x0, 0x800000, PRGMEM;
    mem2 = 0x800000, 0x800000, FREMEM;
    iomem = 0x1000000, 0x1000, IOPORT;
    :
};
virmem { /* 仮想空間構成の定義 */
    :
    u2  = constant, 0x20000000;
    :
};
loadaddr { /* ロードアドレスの定義 */
    krnaddr = 0x0;
    confaddr = 0x100000;
    modaddr { /* 全モジュールの空間配置定義 */
        mod_A # 1 = 0x200000, 0x40000000;
        :
    };
};
initmodule = mod_A; /* 初期モジュールの定義 */
```

図 2: システム定義の例

physmem定義 システムに実装された物理メモリ領域とその属性を記述する．属性には，ロードモジュールのロード領域(PRGMEM属性)，カーネルが管理する空きメモリ領域(FREMEM属性)，仮想空間へのマッピング指定が必要な特殊用途の領域(IOPORT属性，MAPMEM属性)などがある．

virmem定義 仮想空間の構成を定義する．

loadaddr定義 カーネルや各モジュールのロードアドレスを定義する．特に，modaddr定義では，システムを構成する全モジュールの物理／仮想空間における空間配置を指定する．

initmodule定義 システムが立ち上がって最初に起動されるタスク(初期タスク)が属するモジュールを指定する．

3.2 モジュール定義

モジュールは，ソースプログラムのリンク単位[1]であるユニットと，物理メモリの特定領域をデータとして参照するためのマップメモリから構成される (図 3).

```
module mod_A {  /* モジュール mod_A の定義 */
        export      func_A;
        mapmem {  /* マップメモリの定義 */
                mapio = "iofile.o", "rw", iomem;
        };
        unit unit_1 {  /* ユニットの定義 */
                export      func_A;
                object      "unit_1.o";
                mapmem      mapio;
        };
        :
        inittask {  /* 初期タスクの定義 */
                :
        };
};
```

図 3: モジュール定義の例

export 定義 他のモジュールからの参照を許すシンボルを宣言する.

unit 定義 ユニットを定義する．タスクのテキストやライブラリ関数など，ユニットを構成するファイルの指定 (object 定義), ユニットが参照するマップメモリの指定 (mapmem 定義) をおこなう.

また, モジュールが export 宣言しているシンボルは，そのシンボルが属するユニットで export 宣言しておく.

mapmem 定義 マップメモリを定義する．マップメモリによって, physmem 定義で記述した IOPORT 属性や MAPMEM 属性の物理メモリ領域を，モジュールの仮想空間領域にマッピングすることができる.

I/O ポートや DMA のバッファ領域のための仮想空間領域を確保して，このマップメモリ定義で指定したファイルのデータとしてアクセスすることができる.

inittask 定義 初期タスクを定義する．システム定義で初期モジュールに指定 (initmodule 定義) されたモジュールは，初期タスクを定義しなければならない.

3.3 リソース定義

生成可能なタスクの最大数ほか 20 数種類のリソース定義がおこなえる．カーネルがタスクやイベントフラグの生成時に必要とする管理領域を，リソース定義に従ってあらかじめ確保しておくことにより，これらカーネル資源の動的確保によるオーバヘッドをなくすことができる [2].

4 サポートシステムの処理概要

システム構成定義の 3 種類の定義ファイルとソースプログラムから, ロードモジュール (LM) を得るまでの処理の流れを述べる (図 4). 処理は 3 段階に分かれる.

4.1 各言語処理系による処理

ソースプログラムと, SG 言語で記述されたシステム構成定義を，それぞれのコンパイラで処理する.

ソースプログラムは, 既存のコンパイラを使って, リロケータブルオブジェクトに変換される.

3 種類あるシステム構成定義は，本サポートシステムのツールである SG 言語コンパイラ (apsg) を使って，次の 2 種類のライブラリに変換される.

1. システムライブラリ

   システム定義ファイルとリソース定義ファイルから作られる．システムに 1 つ存在し，その主な内容は，カーネルの資源情報やシステムの空間構成など，モジュールに依存しない情報である.

2. モジュールライブラリ

   システム定義ファイルとモジュール定義ファイルから作られ，各モジュールごとに存在する.

これら各ライブラリファイルは，以後の処理段階における他のサポートツールの入力になる．そして，ライブラリファイルが分離しているため，

[1] リンク単位とは，シンボルの参照関係において閉じているプログラムの集まりであり，リンク・エディタが処理する単位である.

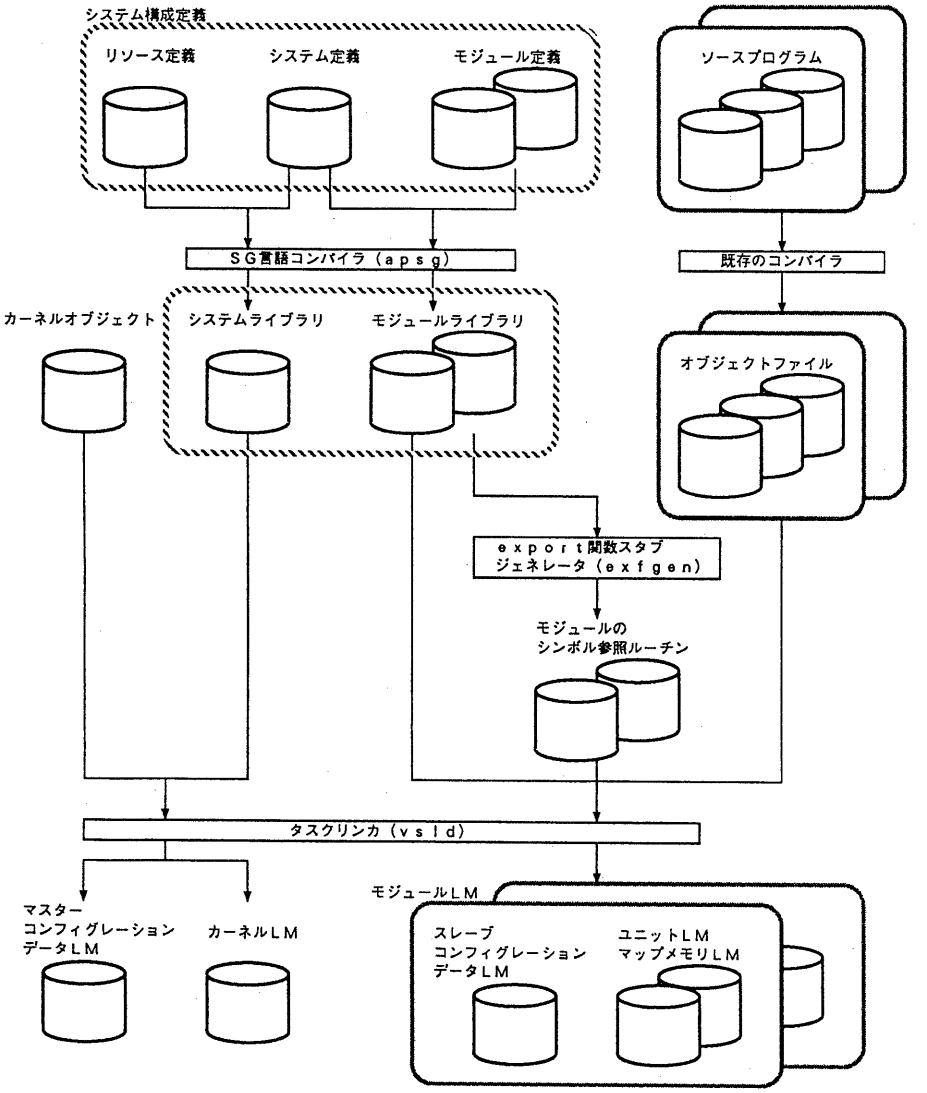

図 4: サポートシステムの処理

あるモジュールが変更されても他のモジュールに影響を与えない.

4.2 モジュール間参照ルーチンの生成

モジュール間で参照できるのは, モジュール定義で export 宣言したシンボルに限られる.

サポートツールの export 関数スタブジェネレータ (exfgen) は, apsg が生成したモジュールライブラリから, このモジュールのシンボル参照に必要なルーチンを生成する.

シンボル参照ルーチンは, export 宣言されたシンボルの値をエントリーとするテーブル (export シンボルテーブル) を介して間接参照をおこなう (図 5).

4.3 ロードモジュールの生成

サポートシステムは, 3 種類のロードモジュール (LM) を, 別々に生成することができる. これらのロードモジュールは, システムライブラ

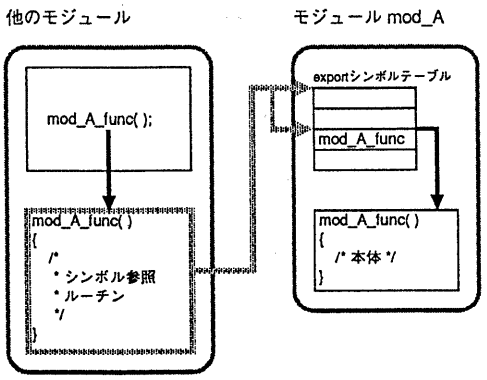

図 5: モジュール間のシンボル参照方法

リやモジュールライブラリの情報をもとに，サポートツールのタスクリンカ (vsld) が生成する．

- カーネル LM

  CTRON 基本 OS カーネルのロードモジュールである．アドレス割付けは，システム定義内の krnaddr 定義に従う．

- マスターコンフィグレーションデータ LM

  システムライブラリの情報 (すなわち，システム定義とリソース定義の内容) をデータにしたもので，システムに 1 つ存在する．アドレス割付けは，システム定義内の confaddr 定義に従う．

- モジュール LM

  そのモジュールに属するユニットやマップメモリ毎に作られる LM ファイルと，モジュールライブラリの情報をデータにしたスレーブコンフィグレーションデータ LM のファイルから成る．

  これらモジュールに属する LM の集まりは，システム定義内の modaddr 定義で指定されたアドレスから，スレーブコンフィグレーションデータ，ユニット，マップメモリの順に割付けられる．

  また，スレーブコンフィグレーションデータには，このモジュールが export 宣言したシンボルの export シンボルテーブルがある．

ここで生成された 2 種類のコンフィグレーションデータ LM は，システムを初期化するためのデータとして使われる．カーネルは，まずマスターコンフィグレーションデータの情報をもとに，物理メモリの実装チェックや，仮想空間管理などの各種管理データ領域の確保と初期化をおこなう．そして次に，モジュールごとに存在する各スレーブコンフィグレーションデータの情報をもとに，仮想空間領域のプロテクションの設定などモジュールの初期化をおこなう．

5 まとめ

CTRON サポートシステムの開発では，システム構成を分割して定義できる記述言語 (SG 言語) の開発をおこない，システムの基本構成要素であるモジュール毎に開発 / 変更が容易にできる環境をサポートツールとして実現した．

参考文献

[1] Quong, R.W. et al : Linking Programs Incrementally, ACM Transaction on Programming Languages and Systems, Vol.13, No.1, Jan., 1991.

[2] 高橋 他 ： V60 リアルタイム OS のソフト開発サポート情報処理学会 第 34 回 全国大会講演論文集 1987.

[3] 内田 他 : V70/V80 CTRON カーネルの実現情報処理学会 第 39 回 全国大会講演論文集 1989 .

[4] トロン協会 - CTRON 専門委員会編 : 原典 CTRON 体系 2 カーネルインタフェース, 1988 .