情報家電に向けた、ソフトウェア部品の開発

Development Software Components for the Internet Appliance

邑中　雅樹 [1)]
Masaki Muranaka

1) 合資会社　もなみソフトウェア
（〒229-1103 神奈川県相模原市橋本５－１７－１３－４０５ E-mail: monaka@monami-software.com ）

ABSTRACT. By progress of semiconductor technology, ubiduitous computing (computing everyware) environment will be realized. It is impotant to connect various embedded systems in the next world. Distributed Object Technolody is considered as a one of the leading technolody. The purpose of this project is to designate a distributed object platform for the embedded environment under limited resources.

1　背景

半導体技術の発展により生活環境のあらゆるところにコンピュータが使われる ubiquitous computing 環境が実現されつつある。そのよう環境においては、数多くの小規模な組込みシステムがネットワークに接続され、協調動作する。多種類の組込みシステムを、効率よくまた高い品質で実現するためのアプローチの一つとして、分散オブジェクトプラットフォームを用いる方法が有力である。

2　目的

本開発の目的は、限られた資源の元で、リアルタイム OS と Java をベースとした分散オブジェクトプラットフォームを構築することである。具体的には、μ ITRON 仕様 OS、TCP/IP プロトコルスタック、JavaVM、HORB をベースとした ORB を含むプラットフォームを、ワンチップマイコンを用いた極めて限られた資源のシステムで動作できるように実装した。

3　開発の内容

3.1　WabaVM の TOPPERS/JSP への最適化

WabaVM を TOPPERS/JSP へ移植した。

WabaVM は移植性を重視して設計されている。それでも、TOPPERS/JSP カーネルへの移植にはいくつかの変更が必要となる。変更点は、WabaVM が PDA 用 OS での使用を想定しているのに対して、TOPPERS/JSP は小規模組み込み用途を想定している、ということに起因するものである。

3.1.1　主な制限

典型的な WabaVM 実行環境と比較して、本案件が想定するターゲットボードの、具体的な制限は、以下のようなものである。

- 可変長メモリ取得の手段がない。
- ファイルシステムが存在しない。
- 描画機能がない。
- カタログやレジストリといった、データベースサポートがない。

可変長メモリ取得以外の要素は、VM 本体に関連するものではなく、ネイティブメソッドの実装に関するものといえる。

実装においては、以下に述べるような変更を加えることで、制限を回避した。

3.1.2　クラスローダの変更

通常の Waba クラスは、Warp 形式と呼ばれるアーカイブ形式にまとめられ、ローカルマシン上のファイルシステムに置かれている。通常の WabaVM は起動時にクラスローダを分析して、必要なクラスを抽出する。

本案件のシステムには、ファイルシステムが存在しないため、static 領域のバイト配列として、Warp 形式のクラスアーカイブを保持させた。WabaVM は、Waba クラスファイルへの書き出しを行わないため、この方式で十分実用に供することができる。

3.1.3　メモリアロケータの変更

WabaVM が仮定する数少ない前提として、下位の OS が、malloc like な可変長メモリ取得 API を持つ、というものがある。しかし、実際に可変長メモリ取得 API を呼び出しているのは、起動時のみで、多くの JavaVM が行うような、ヒープの追加取得は行わない。よって、ヒープを固定長配列から取るよう改変することも、可能である。

可変長メモリ取得により VM が管理する要素を以下に示す[*1]。

vmstack	VM スタック
nmstack	ネイティブメソッド用スタック
class_heap	クラス用ヒープ
object_heap	オブジェクト用ヒープ

WabaVM は、GNU autoconf が生成する configure スクリプトでビルドオプションを定義するが、configure スクリプトに以下のようなオプションを実装することで、固定長配列をヒープとして利用することを指定するよう改変を行った。

[*1] これ以外にも、ネイティブスタック内で、独自に可変長メモリ取得を行う可能性はある。しかしながら、現在 WabaVM が標準でサポートしているネイティブメソッドには、そのようなコードは含まれていない。

–enable-fixed-memory-size Avoid dynamic memory allocation
–enable-fixed-vmstack-size Byte size of fixed vmstack
–enable-fixed-nmstack-size Byte size of fixed nmstack
–enable-fixed-class-heap-size Byte size of fixed class heap
–enable-fixed-object-heap-size Byte size of fixed object heap

3.1.4 機能の無効化

ターゲットボードや下位 OS がサポートしていない機能は、用いることはできず、領域の無駄である。そこで、想定ターゲットに存在しないと思われる機能を無効化できるよう改変を行った。無効化は、機能単位で指定することができる。機能単位は、ほぼ、ネイティブメソッドを含むクラスと同義である。

これらの機能の無効化も、メモリアロケータと同様の手順で行う。具体的には、configure スクリプトから機能の指定を行う。configure スクリプトに付加した、オプションの一覧を、以下に挙げる。

–enable-catalog enable native package for catalog class
–enable-file enable native package for file class
–enable-fontmetrics enable native package for fontmetrics class
–enable-graphics enable native package for graphics class
–enable-gui enable native package for gui(window) class
–enable-serialport enable native package for serialport class
–enable-socket enable native package for socket class
–enable-soundclip enable native package for soundclip class
–enable-thread enable native package for thread class
–enable-registry enable native package for registry class
–enable-wabajtron enable the use of the org.wabajtron classes

3.1.5 効果

–enable-wabajtron を有効に、それ以外を全て無効にした場合、クラスファイルの量は、全て有効にした場合の約 1/6 に圧縮できた。

3.2 JTRON1.0 仕様準拠 API の WabaVM に対する実装

WabaVM と TOPPERS/JSP が制御するタスクの通信用に、JTRON1.0 仕様と同様のアタッチクラスを作成した。

ただし、WabaVM と TOPPERS/JSP の制限により、以下のような変更を行っている。

3.2.1 エラー報告方法の変更

JTRON1.0 仕様では、アタッチクラスからのエラー報告を、例外の送出により行う。しかしながら、WabaVM は、Java の例外に相当する仕様は存在しない。そこで、メソッドの返値にエラー情報を含ませた。

3.2.2 いくつかのアタッチクラスの未実装

μ ITRON3.0 仕様では、レベル E [*2]とされている、可変長メモリプール/ポート/メッセージバッファといった機能へのアタッチクラスが、JTRON1.0 仕様で規定されている。

しかしながら、μ ITRON3.0 仕様における' レベル E' 相当のサービスコールは、μ ITRON4.0 仕様では、スタンダードプロファイル外として規定されている。スタンダードプロファイル準拠である TOPPERS/JSP においても、実装は行われていない。そこで、アタッチクラスのうち、スタンダードプロファイル外のサービスコールに対応するものについては、未実装とした。

3.3 WabaVM のスレッド拡張

SourceForge 版の WabaVM では、Thread クラスへの拡張が行われている。WabaVM におけるスレッドは、JavaVM におけるスレッドとは異なっている。以下に、Waba スレッドの特徴と、どのように実装したのかを示す。

3.3.1 Waba スレッドの特徴

Wabs スレッドの特徴は、以下のようなものである。

- スレッドは、内部タイマーによって、Thread クラスの run メソッドが呼び出される。
- Waba の Thread クラスは、run メソッドからリターンすることで、次のスレッドへの実行権限を与える。(Java の Runnable インタフェースとは異なる)
- Thread.sleep(int) メソッドは、run メソッドからリターンした後、次回の run メソッド呼び出しまでの間隔を指定する。
- スレッドスタックが存在しない。

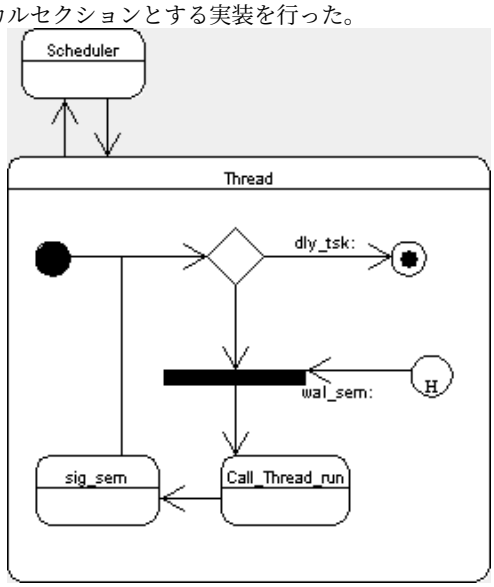
以上をまとめると、WabaVM におけるスレッドは、μ ITRON 仕様でいうところのタスクではなく、周期ハンドラと、類似の実装となっている。プリエンプティブスケジュールを行うことはできない。

3.3.2 実装

本実装では、WabaVM の仕様に準じるような実装方針を取った。しかし、リアルタイム OS である特徴を生かして、実行効率のよい実装を狙った。

現時点の WabaVM の実装では、スレッドの実行間隔を空ループによって計算している。この実装は、言うまでも無く、CPU 資源や消費電力の点で問題がある。

そこで、Waba スレッドと μ ITRON 仕様のタスクを 1 対 1 に対応させ、run メソッドの呼び出し部分をクリティカルセクションとする実装を行った。



複数の実行可能状態のスレッドのうち、run メソッドを実行できるのは、(先にクリティカルセクションに到着した)1 つのスレッドのみである。

*2 拡張機能

3.4 組込向け TCP/IP プロトコルスタックの開発

TOPPERS/JSP に最適化した TCP/IP プロトコルスタックを作成した。実装において、まず留意したのは、‘移植性の確保’、次に ‘省メモリ’、最後に処理速度と拡張性である。

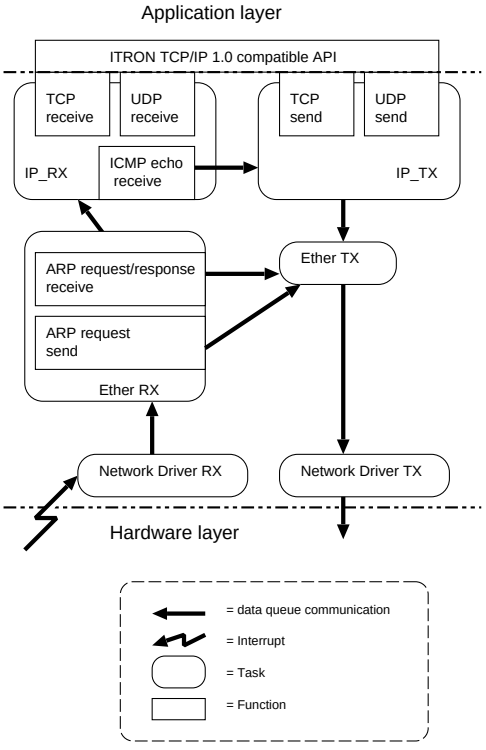
当初、コードのベースとして、大石氏が Web ページ[*3]で公開しているものを、ほぼそのまま移植する予定であったが、内容を解析するうちに、

- スタンダードプロファイル外のサービスコール (特に mbf 系) がある
- 特定の CPU に依存したコードが含まれる
- タスクを分割しすぎて、パフォーマンスが悪い
- 特定のアプリケーションを想定した構成で、API が存在しない。

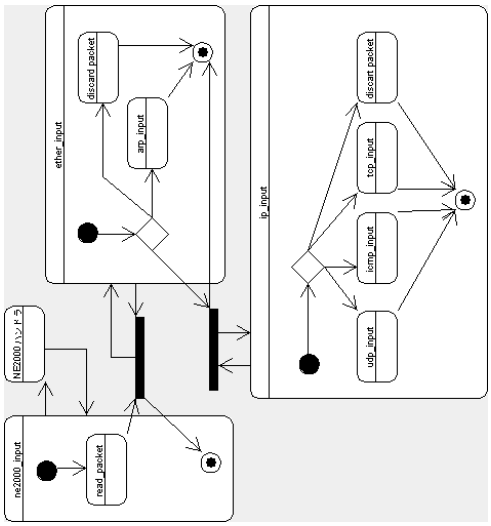
など、本プロトコルスタックの実装上の留意点とは、相反する部分があることが明らかになってきた。そのため、最終成果物においては、大部分を新規に書き直してある。

3.4.1 全体の構成

本プロトコルスタックの全体構成図を示す。



以下に、タスクを跨いだ状態遷移図を掲げる。



3.4.2 実装上の特徴点

本プロトコルスタックの構成により得られる実装上の特徴点を以下に述べる。

3.4.3 省メモリ

なによりもコンパクトであることが、組込用ソフトウェア部品には求められる。本プロトコルスタックにおいて、ライブラリのサイズは 44.7KB、全機能と動作検証用のサンプルを TOPPERS/JSP とリンクしてもイメージサイズ 60KB 以下[*4]を実現した。

3.4.4 移植性の確保

多種のプロセッサが存在する組み込み用ソフトウェア部品において、移植性は重要な要素である。

一般に移植性を阻害する要因は、他のソフトウェアモジュールへの依存と、特定ハードウェアへの依存がある。プロトコルスタックにおいて、考えられる具体的な要因は、下位の OS への依存、ヘッダ内のネットワークバイトオーダと CPU バイトオーダの不一致が、考えられる。

本プロトコルスタックでは、内部で使用するサービスコールとして、μ ITRON4.0 仕様スタンダードプロファイルが指定するもののみを用いた。また、ヘッダ処理において、ネットワークバイトオーダと CPU バイトオーダとの相互変換を行うルーチンを用意し、このルーチンを常に使用することにした。

結果として、μ ITRON4.0 仕様 OS スタンダードプロファイルに準拠していれば、CPU を気にすることなく利用することができる実装となった。

余談であるが、本プロトコルスタックが使用しているサービスコールで、自動車制御用プロファイルに含まれないものは、固定長メモリプール関連のみである。実証はおこなっていないが、自動車制御用のプロトコルスタックとしても利用できる可能性がある。

3.4.5 データコピー回数の削減

プロトコルスタック内部でのコピー回数を削減し、low latency を狙うため、構造体を用意し、Ethernet パケット単位で管理する実装とした。API の直前まで、関数間で受け渡される情報、またタスク間で受け渡される情報は、パケット構造体へのポインタのみである。データは、受信時は Ether ドライバ内部で、送信時には API ルーチンの内部で、パケット構造体に格納される。

結果として、ping のターンアラウンドタイム[*5]500us 以下という性能を得ることができた。

[*3] http://cwaweb.bai.ne.jp/ ohishi/

[*4] i386-elf-gcc version2.95.3 での値

[*5] 10Mbps LAN、ターゲットマシン Pentium(133MHz/ISA bus)、計測マシン PentiumIII(850MHz×2 PCI bus)

3.4.6 プロトコルに対するスケーラビリティ

プロトコルは、時代によって変化している。とくに、数年以内に IPv4 から IPv6 への移行が予測される。そこで、ある程度の冗長を無視しても、プロトコルに対するスケーラビリティを確保するような実装を行った。

各プロトコルの処理は、関数レベルで完全に分離されている。よって、ディスパッチャに書き加えることで、新たなプロトコルに対応させることができる[*6]。また、将来、ネットワーク層/トランスポート層を付加した場合でも、全体のパフォーマンスが低下しないよう、Ether や IP の送受信を、それぞれ別タスクとして設計した。

ネットワークインタフェースドライバも、Ether とは別のタスクとして分離している。これは、より多くの NIC [*7] に対応できるようにするためである。本案件開始時は、SH7615 オンチップ Ether を想定していたが、最終的には ISA バス上の NE2000 での動作確認を行った[*8]。

3.5 ITRON TCP/IP 仕様 like API の開発

上記プロトコルスタックを用いた、ITRON TCP/IP 1.0 仕様互換の API を開発した。ただし、μ ITRON4.0 仕様スタンダードプロファイルを参考に、仕様の変更を行っている。

3.5.1 API の追加

μ ITRON 仕様では、機能を代表する API に加えて、タイムアウトやポーリングの動作に特化した API が存在する[*9]。しかしながら、ITRON TCP/IP 1.0 仕様では、タイムアウトやポーリングの動作に特化した API が存在しない。小規模システム作成時には、動作が特化した API を選択的に使用することで、占有メモリの削減が望める。

実際に付加した API の一覧を挙げる。

tcp_snd_dat	TCP 送信 (完了までブロック)
tcp_tsnd_dat	TCP 送信 (タイムアウトつき)
udp_snd_dat	UDP 送信 (送信完了までブロック)
udp_psnd_dat	UDP 送信 (ポーリング)
udp_tsnd_dat	UDP 送信 (タイムアウトつき)
udp_rcv_dat	UDP 受信 (完了までブロック)
udp_prcv_dat	UDP 受信 (ポーリング)
udp_trcv_dat	UDP 受信 (タイムアウトつき)

3.5.2 TOPPERS/JSP の改変

ITRON TCP/IP 1.0 仕様では、API はエラーコードもしくは正の整数を返すが、μ ITRON4.0 仕様スタンダードプロファイルには、ER_UINT 型の整数を返すサービスコールは定義されていない。このため、TOPPERS/JSP にも、ER_UINT 型を返すサービスコールのためのコードは、存在しない。

ソースコードの見通しを確保するため、例外的に TOPPERS/JSP の改変を行った。変更の具体的内容は、関数 1 つ、C 言語で 5 行分の付加である。

3.6 HORB2.1 下位互換サーバの開発

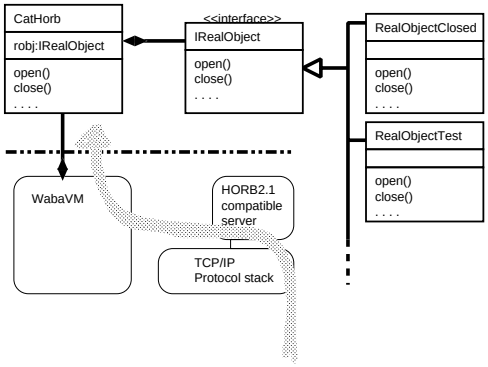
上記開発によって構成されるネットワーク対応組込環境を用い、HORB2.1 と通信可能なフレームワークを作成した。

3.6.1 実装方針

実装においては、サーバとしてのみ動作させる、オブジェクトの動的生成は行わない。提供する Proxy クラスを決め打ちする、ということをサブセット化の指針とした。

3.6.2 実装の概略

また、Waba スレッドは、Java スレッドと大きく異なるため、Waba 上でサーバを実現することは難しい。そこで、サーバ本体は ITRON タスクとし、クライアントからの要求である HORB プロトコルバイト列を解析して、対応する Waba クラスを呼び出すこととした。実装の概略図を以下に示す。



3.6.3 インタフェース

HORB には、オブジェクトをネットワーク透過に扱えること、言語中立になり得るプロトコルが存在すること[*10]、の 2 つの特徴がある。一般には、前者を特徴と捕らえるが、本システムでは、後者に着目した。本システムでは、汎用と思われるクラスを一つ定義し、そのインタフェースのみをサーバ経由で外部に公開する実装とした。インタフェースが公開するメソッドは、平成 12 年度 IPA 未踏ソフトウェア創造事業の成果である、Neko-bus での経験を活かしている。Java 言語で記述したインタフェースを、以下に掲げる

```
public interface ICatHorb {
 public int openObject(String n /* object name */);
 public int closeObject();
 public int readRecord(byte[] buf, int len, int off, int rn);
 public int writeRecord(byte[] buf, int len, int off, int rn);
 public int makeRecord(byte[] buf, int len, int off, int rn);
 public int removeRecord(int rn);
}
```

3.6.4 実装とコードサイズ

上記のインタフェースを実現するコードを実装した。クラスファイルが約 4.5KB、サーバとなる ITRON タスク部約 7.5KB、合計 12KB に収めることができた。

4 今後の課題

本システムでは、小規模に作成するという前提があり、インタフェースを 1 つに絞り込んだ。結果として組込機器とより大規模な機器とののアクセスを簡略化する、統一的なインタフェースを実現した。しかしながら、メモリと機能のトレードオフをもうすこし機能寄りにして、より柔軟な環境にすることで、組み込み向け ORB を望むユーザ層に訴求するものになりえたかもしれない。オブジェクト転送を含まないなどの制限を許容すれば、50KB 以内に収まる実装も可能であるという感触を得ている。柔軟なインタフェースの提供が、今後の課題である。

[*6] ソースコード上では、ip.c や ether.c に対応する

[*7] Network Interface Controller

[*8] 計画変更承認申請済み

[*9] snd_dtq に対する tsnd_dtq や psnd_dtq が相当する。

[*10] HORB には、Java 版以外にも、非公式ながら C/C++/Delphi 版が存在し、相互接続が確保されている。