JTRON仕様の実装とその現状

(株)アプリックス　根本　忍

はじめに

組込み機器が，徐々にネットワーク機能を備えるようになってきている．これらの機器を協調動作させることで超機能分散システム[1]の実現に近づくことができるが，そのためにはネットワークに対応したリアルタイムOS（RTOS）の仕様が必要になる．それがJTRON（Java Technology on ITRON）仕様である．本稿では，まずJTRON仕様の概略を紹介し，実装例とその現状について述べる．また，Javaでリアルタイム処理やハードウェア制御を行う際のいくつかの問題点について論じ，JTRON仕様でそれらをどう解決しているかを説明する．

JTRONとは

JTRONは，すでに10年以上の実績を持つリアルタイムOS仕様のITRON[2]と，移植性やネットワーク透過性に優れたJavaの実行環境とを融合した仕様である．JTRON仕様を使った応用システムでは，ITRON仕様とJavaの両方の長所を活かした開発が容易に行える．具体的にいえば，時間的制約の厳しいリアルタイム制御プログラムはITRON仕様OSの機能を使って記述し，GUIやネットワーク関連機能はJavaで記述することになる．これにより，ITRON仕様とJavaを融合させたネットワーク対応のリアルタイムシステムが構築できる他，性能面でのチューニングが必要な部分をITRON仕様OS側でネイティブコードを使って記述し，移植性を重視する部分をJava言語で記述してPCやWSを利用した開発・デバッグを行うこともできる．JTRONの仕様としては，1997年12月にJTRON1.0仕様が，1998年10月にJTRON2.0仕様が，それぞれ公開されている．JTRON仕様に関する詳細情報や仕様は，TRONプロジェクトのWebサイト[3]から入手することができる．JavaおよびJavaの実行環境については，参考文献4），5）に詳しい．

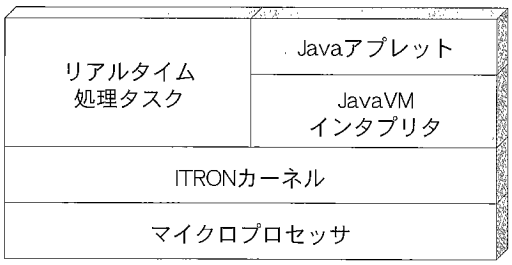
JTRON仕様の設計

ITRON仕様の設計は，これまでタスクのスケジューリングや同期通信機能などといったリアルタイムカーネルの部分を中心としており，ネットワーク機能やGUIについてはインプリメント依存となっていた．最近ではITRONをターゲットにした組込み向けのTCP/IP仕様やITRON向きのGUI仕様（ITRON-GUI）も標準化されつつあるが，いずれもITRONの特長である「組込みシステムへの適応化」「弱い標準化」の方針を具体化したものであり，たとえばオブジェクトコードの互換性までを狙ったものではない．

一方，ネットワークに相互接続され協調動作するような組込み機器（インテリジェントオブジェクト[1]）を実現するためには，プログラムモジュールを実際にダウンロードして実行するケースも考えられる．このような場合には，Javaの特質とするような，強く標準化された実行環境が必要となる．

JTRON仕様のプラットフォームは，こういった場合に有効である．つまり，ネットワークを通じて協調動作するような機器では，ネットワーク関連機能のサポートとプログラムの互換性が重要な課題となるが，この2点を大きな特長としたJavaに注目し，それを取り入れた新しいリアルタイムOS（RTOS）の仕様としてITRONを拡張したものが，JTRON仕様である．

JTRON仕様では，Javaの言語仕様の特質を損ねることなくリアルタイム制御APIが設計されている．特に，Javaが言語レベルで備えているマルチスレッド機能やモニタでの同期のサポートと，ITRON仕様が持つ同期・通信サービスをJavaでも利用できるようにする点を重視している．このために，Javaのスレッドと ITRONのタスクとを1対1でマッピングし，すべてのコンテキストをITRON仕様カーネルのスケジューラで実行するように設計されている．総じて，Javaの持っている機能とITRON仕様OSの機能とを合わせられる部分においては，その機能を現状のJavaの機能として使用できるように設計し，現状のJavaが提供している機能だけでは設計できない部分に関しては，できる限りJavaの特質を失わないように設計し新たに組み込むことが想定されている．このため，Javaプログラマが容



「Javaアプレット＋JavaVMインタプリタ」も
「リアルタイム処理タスク」もITRONのタスクとして動作する.

図-1　JTRONアーキテクチャ

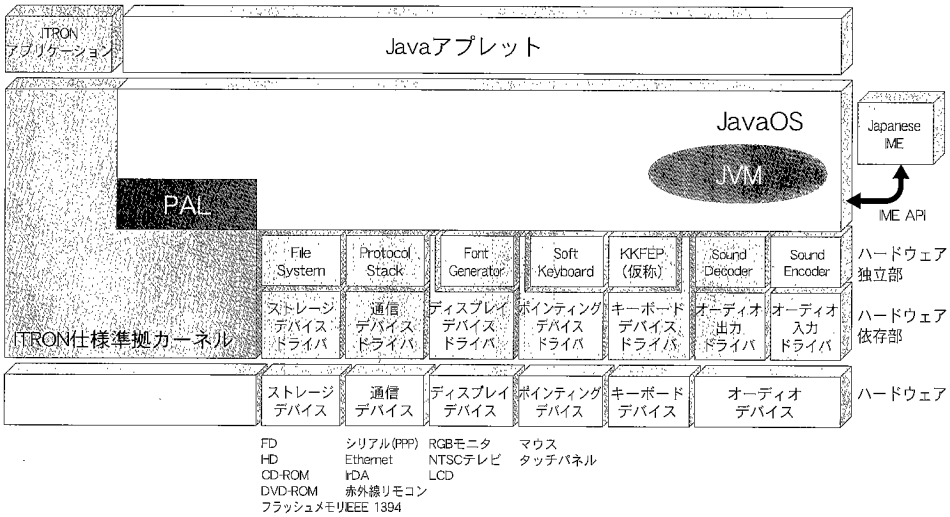


図-2　JBlend構成図

易にJTRON仕様アプリケーションを記述できるようになっている.

また，ITRONのリアルタイム実行環境にはまったく手を加えずに，その上にJavaの実行環境を実現していることもJTRONの特長となっている（図-1）．つまり，Javaの実行環境はITRON仕様OSのアプリケーションタスクとして動作する．すなわち，Javaの実行環境を構築するためのカーネルとして，ITRON仕様カーネルを用いるのである．この方法では，ITRON仕様カーネルおよびそのアプリケーションタスクの動作環境がJavaの実行環境の制約を受けないため，ITRON仕様の最大の特長の1つであるリアルタイム性を損なうことがない．先にも触れた通り，JTRON仕様ではJavaのスレッドがITRON仕様OSのタスクとして実装されている．Javaのサポートするマルチスレッド機能を利用することで，機器制御などを行うマルチタスクプログラムの開発が容易になる.

JTRON仕様の実装例

JTRON仕様の実装は，特定のITRON仕様OSやJavaの実行環境に限定されるものではない．すでに出荷されている製品としては，以下の2つがある.

■実装例（1）JBlend

アプリックス社の「JBlend」[6]では，各社から提供されているITRON仕様OSとサン・マイクロシステムズ社から提供されているJavaOSとを，併存的に構成している．実装を行ったITRON仕様としては，ITRON1，μITRON2.0，μITRON3.0の3種類があり，ITRON仕様OSには手を加える必要がない．JavaOS[7]は，Javaの実行に最適化されたプラットフォームで，ホストオペレーティングシステムを必要とせずに，ハードウェアのプラットフォーム上でJavaアプリケーションを直接実行できる環境を提供している．現在のJBlendに実装されているのは，JDK1.1.3ベースのJavaOSである（図-2）.

JBlendにおける実装では，ITRON仕様カーネルとJavaOSが接する部分に緩衝層（PAL: Processor Abstraction Layer）を設け，PALとJavaOSとのイン

タフェースはJavaOSで使われている関数名や引数の型を継承し，PALとITRON仕様カーネルとのインタフェースにはITRON仕様が提供するインタフェースをそのまま利用している．リアルタイム機能については，JavaOSが行っていたスレッドの生成や削除，優先順位の管理，割込みと排他処理，イベントモニタなどの機能を，ITRON仕様カーネルの機能を使って書き換えることで実現している．メモリ管理については，ITRON仕様カーネル側で管理する領域とJava側で管理する領域とに分割している．ITRON仕様カーネル側で管理するメモリ領域は，イベントフラグやセマフォ，メールボックスなどに使われる．Java側で管理する領域は，Java実行環境のヒープとして使用され，この領域はガベジコレクション（GC）の対象となる．JBlendは，現在，日立製作所のSuperH（SH）ファミリをはじめ，日本電気V800およびVRシリーズ，東芝TX39，富士通SPARClite，Intel StrongARM（SA-1100），各種8086互換CPUなどの複数のプロセッサに移植されている．

■実装例（2）J-right/V

パーソナルメディア社の「J-right/V」[8]は，PCで動作するμITRON3.0仕様カーネルの上にJavaOSを実装している．J-right/Vに含まれているμITRON3.0仕様準拠OS「I-right/VJ」は，システムコールのレベルE（拡張機能）までをサポートしている．もともとJavaOSは，NC（Network Computer）用として開発されているため，ローカルストレージのサポートがない．J-right/Vにおいては，通常のPCとしても使用できるようにローカルストレージのサポートが行われている．また，PC用のさまざまな周辺デバイスにも今後対応させやすいように，デバイスドライバはITRON上で（ITRONのアプリケーションとして）実現する構造となっている．

なお，J-right/Vは，前出のJBlendをパーソナルメディア社とアプリックス社が共同でPCアーキテクチャに対応させた製品である．

各種リアルタイムJava仕様とJTRONとの比較

1998年末から，Java自体をリアルタイム拡張しようという動きがでている．標準化をめざすワーキンググループが登場しているが，これらのグループの活動とJTRON仕様との関係について説明する．

1998年11月2日にアナウンスされたReal-Time Java Working Group[9]は，もともとNIST（National Institute of Standards and Technology）[10]が後援するJava言語リアルタイム化要求仕様定義グループから派生したものだ．この他に，複数のRTOSベンダがJava Virtual Machine（JVM）を開発して自社RTOSに搭載するという動きも出てきている．Real-Time Java Working Groupでは，JVMの内部にインクリメンタルに処理が実行されるリアルタイムGCを実装しようとしているが，実際の産業界で要求される高い信頼性と厳しい実時間応答性を達成するには，まだ実績が不足していると考えられる．NISTの動きでは，独自のJavaの拡張が増えることになりプラットホーム独立性が揺らぐことにもつながりかねない．以下では，これらのJavaリアルタイム拡張仕様と比較する意味も含め，Javaのリアルタイム化とJTRON仕様におけるリアルタイム処理について述べる．

Javaのリアルタイム化

Javaのリアルタイム化には，ガベジコレクション（GC）とハードウェア制御の2つの要素が絡む．Javaの場合，変数を含めたすべてのオブジェクトは，原則として生成された時にヒープから確保される．この確保された領域は明示的には解放されないため，ヒープのフラグメンテーションが起きた時にGCが発生する．また，サン・マイクロシステムズ社から提供されている現在のJavaの実行環境では，割込みなどの要因によりGC中に非同期イベント処理を行う必要が生じても，GCが終了するまでその処理は待たされてしまう．これは，GCが完了するまですべてのスレッドのスケジューリングがロックされていることによる．この問題を解決するために，HPをはじめ各社から提供されているリアルタイム性の高いJavaの実行環境では，GCの中断を行ってスレッドの再スケジューリングを行える実装にし，GC中に発生した非同期イベントを扱うことが可能になっている．こうした中断可能なGCアルゴリズムの実装自体は比較的容易だが，相当注意して実装しないとGCの実行効率が落ち，全体のパフォーマンスが低くなってしまう．

中断可能なGCにおける問題点

中断可能なGCアルゴリズムには，フラグメンテーションの解消効率が悪く，メモリ消費量の予測が困難になるものがある．GCを中断して割込み可能なJava実行環境を評価する時は，割込み応答性や応答時間の予見可能性以上に，GCの実行速度やメモリ使用効率に注意しなければならない．中断可能なGCについては，Java仕様の変更やプログラミング方法の変更を伴わないので，Java実行環境を提供する各団体や企業の実装を待てばよいようにも思えるが，GCについてはもう1つ課題がある．それは，GCを中断している間の非同期イベント処理中におけるメモリの扱いである．Javaでは，メソッド内のローカル変数を含め，原則としてすべてのオブジェクトがヒープより確保される．この場合，メソ

ッドを呼び出すたびにオブジェクトが生成され，GCが発生する可能性がある．仮に割込みによってGCが中断できても，割込みをサービスするメソッドが呼び出されたとたんGCが起きるようでは，結果的に応答速度を悪化させることになる．これに対して，割込みをサービスするメソッドが使用するオブジェクトをあらかじめ生成しておくか，非同期イベント専用のヒープを用意して対応する方法が考えられる．あらかじめオブジェクトを生成する場合，メソッドの中で呼んでいる他のメソッドまでプログラマが注意する必要があるが，Javaの構造上これは非常に困難であり，検出の難しい異常な動作を無意識に折り込んでしまう危険性が拭えない．また，非同期イベント専用のヒープを準備する場合，なんらかの方法でこのヒープの仕様を指定しておく必要があり，Javaの仕様を拡張する必要が出てくる．中断可能なGCには，以上で述べたような実用上の問題点が多く，これだけでシステム全体での「リアルタイム処理」ができるとは限らない．

これに対して，JTRON仕様のリアルタイム処理ではまったくアプローチが異なっており，GC自体もITRONから見ればアプリケーションタスクの1つに過ぎない．JTRON仕様では，リアルタイム処理の必要なITRONアプリケーション側の環境と，GCを含めたJava側の環境とを分離し，前者の優先度を高めて実行することにより，GCにまったく影響されないリアルタイム処理が可能となる．

JTRON仕様におけるハードウェアの制御

リアルタイムシステムの場合，割込み応答性が要求されるのはハードウェアイベントである．GCの問題を解決しても，ハードウェアイベントを受け付けられないと意味を持たない．この場合の対処方法は，2通り考えられる．1つは，Javaの仕様を拡張しハードウェアを直接制御できるような方法を提供するもの，もう1つは，ネイティブコードでハードウェア割込みを処理しその情報をJavaのメソッドに通知するものである．前者の方法ではCPUに強く依存してしまうことになる．まず，Javaのメソッドが受け付ける割込み要因とその優先度を指定する必要がある．これはCPUの種類によって，大きく異なる．さらに，割込み発生後直ちにJavaのメソッドが呼び出されるため，CPUの割込みコンテクスト内に存在することになるので，割込みマスク制御をする必要がある．この時点で，多重割込み禁止やより優先度の高い割込みの許可など，CPUや割込みコントローラの構造に依存する処理が発生する．この他，ハードウェアを直接制御するためにポートやメモリ空間を直接読み書きする方法も必要になる．そのためには，Javaにはアドレスの概念がないこともあり，なんらかの拡張をすることになる．この対処方法は，Javaのプラットフォーム独立という基本理念を大きく変えてしまうことになるだけでなく，機種依存部分がJavaで記述されているため，PCやWSを利用したクロス開発が困難になり，ターゲット上にJavaに対応した開発環境を搭載する必要性に迫られることになる．しかし，多くの家電機器や産業機器では，メモリや二次記憶装置などのリソースが限られており，当然ながらキーボードなどは用意されていない．表示装置があるとしても，ターゲットアプリケーションによって占有されている上に表示領域が小さいことから，開発環境の搭載を想定しにくい．またこの方法では，割込みに対する処理がJavaのバイトコードで実行されるため，よりクリティカルなレスポンスが要求されるハードリアルタイムを実現することは困難である．

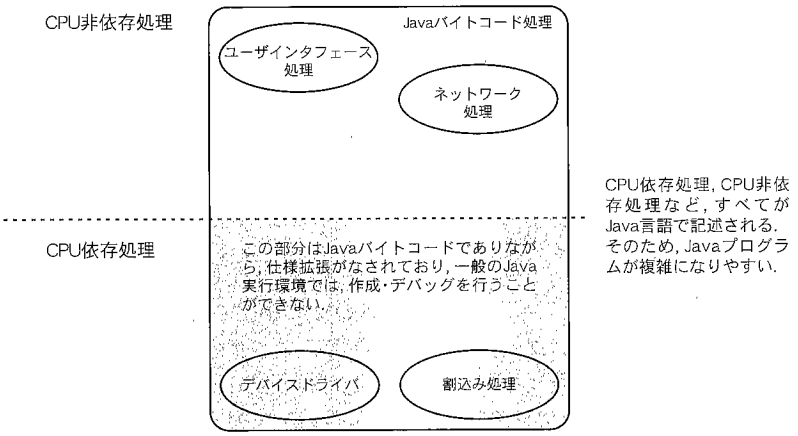
以上に対して後者の方法の場合，ネイティブコードで書かれているため高速処理が可能なだけでなく，CPUに依存した処理もすべて行うことができる．また，その結果だけをイベントやデータとしてJavaのメソッドに渡すことができるため，CPUに依存するようなJavaの仕様拡張をする必要がない．つまり，CPU独立部がJavaコード，CPU依存部がネイティブコードと明確に整理でき，CPU依存部を局所化できるため，全体の見通しがよくなる．

JTRON仕様においては，後者の方法を用いて，Javaのプラットフォーム独立というメリットを保ちながらリアルタイム性を実現している（図-3）．

JTRONの今後と展望

今までに提供されているJavaの実行環境としては，サン・マイクロシステムズ社から提供される各種製品が主流だが，Javaの標準規格化の動きなどもあり，現在では各社からJVMが提供されているほかオープンソースのものもいくつか登場している．さらに，1998年後半からは市場からの要求に変化が見られる．これは，サン・マイクロシステムズ社から組込み用として提供されているPersonalJavaやEmbeddedJavaといった環境[11), 12)]が，実際の組込み用途に用いるのがそのままでは難しい実装であること，もう1つには市場からの要請がいわゆるフルJava（JDK）と組込み用に完全に特化されたものとに二極化してきており，PersonalJavaなどのようにそれらの中間的な意味あいのもの，つまりJDKから不要と思われるグラフィック機能などを省いてフットプリントを小さくした環境の需要が，若干減少していることなどによるものと思われる．こうしたことから，JTRON仕様に組み込まれるJavaの実行環境も，今後は次第に組込み用途での使用を前提としたものに切り替わっていくと予想される．

リアルタイムJavaの場合



JTRON アーキテクチャの場合

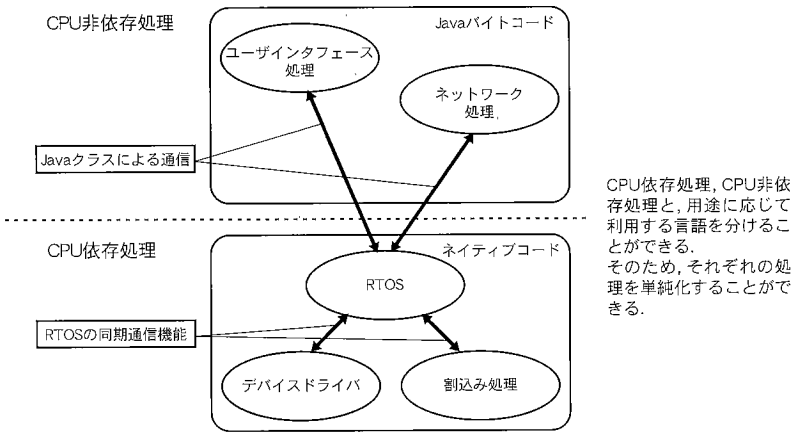


図-3　JTRON仕様におけるリアルタイム処理

おわりに

国際的にも成功したITRON仕様と同様に，JTRON仕様も今後ますます注目され普及するものと予想される．今後は，より小規模の組込み機器やICカードへの対応なども含め，JTRON仕様OSの製品化がさらに進むと思われる．

謝辞　本稿の執筆にあたり，終始ご指導をいただいた東京大学坂村健博士とご協力いただいた関係各位に深謝いたします．

参考文献

1) 坂村　健: TRONの思想と今後，情報処理，Vol.30, No.5（May 1989).
2) 坂村　健編: μITRON3.0標準ハンドブック，パーソナルメディア (1993).
3) TRONプロジェクト: JTRON仕様書, 1997, <http://tron.um.u-tokyo.ac.jp/TRON/JTRON/>
4) 特集Java言語: いまなにが課題なのか，情報処理，Vol.39, No.2（Feb. 1998).
5) サン・マイクロシステムズ: <http://www.java.sun.com/>
6) アプリックス: 1997-1999, <http://www.JBlend.com/>
7) 日本サン・マイクロシステムズ: 1997, <http://www.sun.co.jp/tech/whitepapers/JavaOS/JavaOS.html>
8) パーソナルメディア: 1998, <http://www.personal-media.co.jp/>
9) Real-Time Java Working Group: 1998-, <http://www.newmonics.com/webroot/rtjwg.html>
10) National Institute of Standards and Technology: 1998-, <http://www.nist.gov/rt-java/>
11) サン・マイクロシステムズ: PersonalJava, <http://www.java.sun.com/products/personaljava/>
12) サン・マイクロシステムズ: EmbeddedJava, <http://www.java.sun.com/products/embeddedjava/>

（平成11年2月9日受付）