Vol.2011-OS-116 No.10
2011/1/25

Plan9 と ECMAScript を用いた 分散組込みシステムのプログラミングシステム

盛合 智紀[†1] 佐藤 未来子[†1] 並木 美太郎[†2]

本研究では，分散システムのノードとして組込み機器を用い，アクセス透過性の高い分散システムを構築する手法を提案する．システム全体の計算機資源を単一のファイルツリーとして仮想化することで，アクセス透過性の高いノードの利用が可能である．Plan9 とネットワークファイルプロトコルである 9P プロトコルを用いることで実現する．ノードのプログラミング言語には ECMAScript を用いノードの処理としてバイトコードを制御実行する方法とする．仮想化や通信全般の処理を VM が行うことでプログラマからプロトコルなどの記述によるコーディングの負担を軽減する．ノードをオブジェクトとしてみなすことでシステム全体の記述性が向上し，ノード間の協調動作もアクセス透過性の高い記述が可能である．VM は ROM200KB 程度，RAM が 30KB 程度必要であるが組込み機器のメモリサイズに収まる．実行性能はネイティブコードの 1/30 程度遅く，他の C 言語のインタプリタの 200 倍程度早くなり，センサーノードの処理系として十分な性能となった．

Programming System for Distributed Embedded System using Plan9 and ECMAScript

Tomoki Moriai,[†1] Mikiko Satou[†1]
and Mitaro Namiki[†2]

This paper describes programming system for distributed embedded system with access transparency using TCP/IP network and 9P protocol which is designed for Plan9 operating system. This system supports Remote Procedure Call interface with file I/O interface of Plan9. ECMAScript is used in Language processor on the node. An object in ECMAscript corresponds to a node so that this system considers all nodes as one ECMAscript language processor for embedded systems. A prototype system includes embedded server implemented with Virtual Machine for subset ECMAScript. As the result of prototyping, ROM size is 200KB and RAM size is about 30KB for VM. In the compared execution time of native code and SilentC interpreter, VM marked about 1/30 times slower than native code and VM marked 200 times faster than SilnetC.

1. 背　　景

近年，何処でも高速なネットワークを利用することができるようになりつつある．さらに，近年の組込み機器の高性能化・低価格化も注目を浴びている．これは CPU やメモリのみならず入出力装置等の周辺装置に関してもあてはまることであり，その結果として小型の組込み機器からネットワークに接続できるなど，多種多様な入出力装置を利用できるようになった．

このような，拡充されたネットワークと高性能化した組込み機器の一つの利用例としてセンサーネットワークが有り，地震検知や気温・湿度計測，プラントにおける在庫管理など，設置環境を把握する場面で活用できる．そのため，現在ではセンサーネットワークを構築するための小型組込み機器として，用途毎に様々な物が開発され今後利用される機会が増える．

このようにハードウェアの整備が進むにつれ，センサーネットワークを構築するためのシステムの開発も求められている．センサーネットワークをはじめとした，ネットワークと組込み機器の連携を実現するための一つの手法として分散システムが挙げられる．

2. 問 題 提 起

組込み機器を対象とした分散システムを構築する際の課題を述べる．

(1) 分散システムの資源管理

従来の分散システムの構築方法は大規模なシステムを想定しているものが多く，十分な計算機資源を有した計算機を対象としていた．本研究のように小型の組込み機器を対象とする分散システムで大規模向けの方法を利用することは計算機資源が厳しい．IIOP[1)] 等のプロトコルを提供するミドルウェアが組込み機器上で利用できても，通信待ちや通信内容，組込み機器がどの計算機と通信するかなどを考慮してソースコードを記述する必要が有る．本来記述したい組込み機器の動作を記すソースコード以外に記述する内容が多く存在する．

(2) プログラムの書き込み

分散システム中に多量に存在する組込み機器に対して，距離の離れた設置場所に赴いて直

†1 東京農工大学工学府
Tokyo University of Agriculture and Technology
†2 東京農工大学工学研究院
Tokyo University of Agriculture and Technology

接組込み機器を触りながら，個別の組込み機器毎に用意されたソースコードをチューニングしつつ管理するのは大変な労力を要する．ソースコードの変更の際，組込み機器の数に比例して変更にかかる労力が増えるのも大きな課題である．

3. 先行研究

(1) Arduino[2)]

Processing/Wiring 言語と呼ばれる C,C++に似た言語を用いて開発を行う．コンパイラから生成されるコードは AVR のネイティブコードであり，想定しているハードウェアも特定のものである．シールドと呼ばれる拡張ボードを使用することでイーサーネットを利用することもできるが，イーサーネットに関する記述をソースコード中に記述する必要が有り，MAC アドレスなど個々のノードによって異なる記述を含めたコードを対応するノードに書き込む必要が有る．

(2) Inferno[3)]

Bell 研究所が Plan9[4)] を元に作成した分散 OS である．設計理念や通信プロトコルを含む大部分は Plan9 と共通である．Plan9 との最大の違いは，Inferno が独自に開発した Dis と呼ばれる VM 上で動作していることであり，Dis の機械語を生成する為に，Limbo と呼ばれるプログラミング言語を用いた開発環境を用いる．Dis は Windows や Linux 等をホスト OS として，アプリケーションとして動作させることも，x86 は勿論，ARM や PPC アーキテクチャ上で直接動作させることもできる．Dis はメモリ管理ユニットの無いハードウェアでも動作するとされているが，1MB 程度の RAM が必要になる．小規模のノードには不向きである．

4. 目的

本研究では，組込み機器を対象とした分散システムにおいて，システム全体を透過的にプログラミングできる環境を提供する．本システムではシステム上の計算機資源を利用するものを「クライアント」，計算機資源を提供するものを「サーバ」と呼び，組込み機器を「ノード」と呼ぶこととする．クライアントでもサーバでもプログラマから通信に関する記述を隠蔽し，本来記述したい処理の内容に注力できるようにする．個別のノードにコードを書き込む手間も減らし，開発や運用の労力がノード数に依存しにくくする．

5. 方針

ノード単位でコーディングを行うのではなく，分散システム全体を一つの単位としてコーディングを行う．ノードはオブジェクト指向言語のインタンスに該当し，同じクラスから同じ機能を持つ複数のインタンスを生成する．多くのノードが同じ機能を持ちうる組込み機器を対象とした分散システムでは大きなメリットとなる．ノード毎では無く機能毎にソースコードを管理できるので，ノードの数が増えてもソースコードを管理し易い．「他のノードと通信を行い計算機資源を利用する」ことが「インスタンスのメンバにアクセスする」ことに隠蔽される．また，ネットワーク越しにノードの実行ファイルを転送し，ノードの管理を行う．ノードに実行ファイルを転送するプロトコルとノードの計算機資源を利用するためのプロトコルは，メモリを節約するために同じものを用いる．ノードのコンフィギュレーションに必要な情報はソースコード中に記載されている情報から自動的に生成することで，プログラマの負担を軽減する．

6. 実現方法

分散システムの通信プロトコルに TCP/IP と Plan9 のネットワークファイルシステムの通信プロトコルである 9P[5)] プロトコルを用いることで，アクセス透過性の高い分散システムの構築法を提案する．提案した分散システムを実現するために，数 KB 程度の RAM で動作するスタックマシン型の VM と，ECMAScript のサブセット的な開発言語を用いた処理系を提供する．VM がノードにおける 9P プロトコルのプロトコルスタックを保持し，通信内容や通信の対応をプログラマから隠蔽する．

ノードのコーディングは個々のノードを最小の単位とするのではなく，システム全体を一つのコーディング単位とする．ノードを ECMAScript のオブジェクトに対応付け，メソッドやプロパティをファイルに仮想化する．ノードを超えて他のノードの資源を活用する方法は，通常の ECMAScript におけるオブジェクトの参照やプロトタイプチェーンが対応する．ECMAScript を利用することで，各ノード間の通信プロトコルを始めとした通信に関する記述を隠蔽しつつ，動的に他のノードの計算機資源を利用する記述が可能である．プロトタイプベースである ECMAScript のプロトタイプチェーンのメリットであり，プログラマが親を明示せずに親オブジェクトを遡り続けた探索が可能である．通信に関わる記述がプロパティの参照に隠蔽されており，プログラマはノード間の通信手段を気にせずにアクセス透過性の高い方法で分散システムの計算機資源を利用できる．また，一つのクラスから複数のオ

ブジェクトを生成できるので，ノードに対応するオブジェクトの管理も容易である．プロトタイプベースでは生成済みのオブジェクトに対してプロパティを追加できるので，ノード毎の特徴に合わせて処理の追加や削除も容易である．

本システムにおける通信は全て 9P プロトコルによるファイル入出力に帰着する．つまり，9P プロトコルを利用できればシステムの計算機資源をファイル入出力で利用でき，ノード同士の連携も 9P プロトコルで実現できる．ECMAScript で記述されたオブジェクトやプロパティを 9P プロトコルで呼び出す為に，システム全体が記述された ECMAScript の各要素を一つのファイルツリーに対応付ける．システム全体を表すファイルツリーを各ノードが保持するオブジェクトの部分木に分割して，ノードの持つ計算機資源を活用するためのインタフェースにする．クライアントはリモートマウントでローカル空間にある自身のファイルツリーの一部に，ノードの提供するファイルツリーを接続することでアクセス透過性の高い計算機資源の利用が可能である．システム内の VM 同士が行う通信と，クライアントから送られる通信に同じ 9P プロトコルを用いて組込み機器の計算機資源を節約する．

7. 設　　計

本研究では，システムの計算機資源を利用するものを「ユーザ」，ノードに載っている処理系を「VM」，システム全体をコーディングするものを「プログラマ」と呼ぶ．

7.1 システム構成

図 1 に本システムの概念図を示す．本システムでは同じネットワーク上にユーザもプログラマもノードも存在する．システムに参加するクライアントに要求する要件は 2 点で，TCP/IP 通信のサポートと，9P プロトコルのクライアントとしての機構である．本システムにおいては，9P プロトコルでファイル入出力を発行することにより，クライアント側に新たなミドルウェアを導入せずノードの資源を利用できる．この場合，9P プロトコルを発するクライアントは PC で有る必要は無い．

本研究において，サーバは小型の組込み機器であり，サーバにはスタックマシン型の VM を搭載する．この VM 用の実行ファイルを出力するコンパイラとして，ECMAScript のサブセットを用いる．なお，ファイルに仮想化してクライアントに見せるために仕様を一部拡張している．本 VM が 9P プロトコルスタックを保持しており，クライアントからのファイル入出力や他のノードからのファイル入出力など通信内容を判断して適切な処理を自動的に行う．ECMAScript におけるオブジェクトの参照が 9P プロトコルによる通信に対応する．

7.2 ファイル木に取り込まれた分散ノード

ユーザにとって，ノードはリモートマウントによってローカルファイルツリーの一部にマウントして利用するものである．よって，複数のノードがノード単位で異なる部分木をファイルツリー内で形成することとなる．

一方プログラマもリモートマウントを用いてファイルツリーにノードを取り込むのだが，システム単位でファイルツリーにマウントするイメージになる．本システムではシステム全体がノードの枠を超えて単一のファイルツリーを形成することでプログラマにアクセス透過性の高い開発環境を提供している．実際には，ノードが他のノードをマウントすることでノードを超えてファイルツリーを形成するのだが，ルートノードとなるノードは必ず存在する．ルートノードとはシステム全体で掲載するファイルツリーの中でそれ以上遡ることが

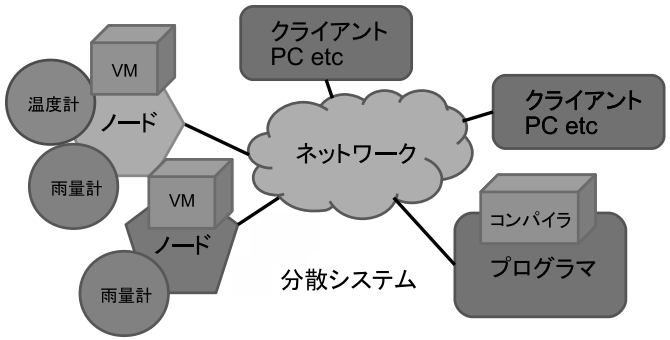

図 1　システム概念図

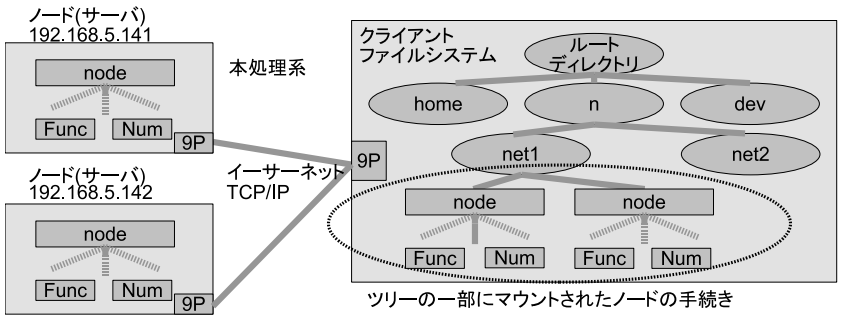

図 2　ユーザから見たオブジェクト

出来ないディレクトリを有するノードのことである．プログラマはルートノードをローカルファイルツリーの一部としてマウントし，そこに各ノードで実行されるユーザコードからコンパイラで生成した実行ファイルを転送することでシステムを構成する．結果として，ルートノードの下に幾多のノードをマウントしたシステム全体を表すファイルツリーが生成され，プログラマのローカル空間に保有するファイルツリーのルートディレクトリがマウントされた先を拡張する．ルートノードから処理すべきコードを受け取った各ノードが，図 2 で示したユーザにマウントされるノードになる．ルートノードも他のノードと同様にユーザのローカルツリーにマウントできる．ただし，ユーザがマウントした場合はルートノードのみが保有する部分木のみマウントされ，他のノードへと繋がるファイルツリーは提供されない．ノード同士が連携してシステム全体で一つのファイルツリーを生成した結果が図 3 である．

図 2，図 3 において，クライアントのファイルシステムの部分木として，サーバの VM に登録されている手続きが含まれる．サーバの提供するファイル木は，サーバ上の VM で動作する命令コードから生成されたものであり，VM の実行を管理するファイルや，プログラムの結果を取得するファイルが入ったオブジェクトの階層である．そこで，上記のファイルに仮想化するための情報として，メソッド名やプロパティ名が別途必要になるので，サーバで動作させる実行ファイルに必要な情報をプログラマがソースコード中に記述することが設定の手続きとなる．コンパイラの支援により，別途プログラマがサーバ上で各種名称の設定

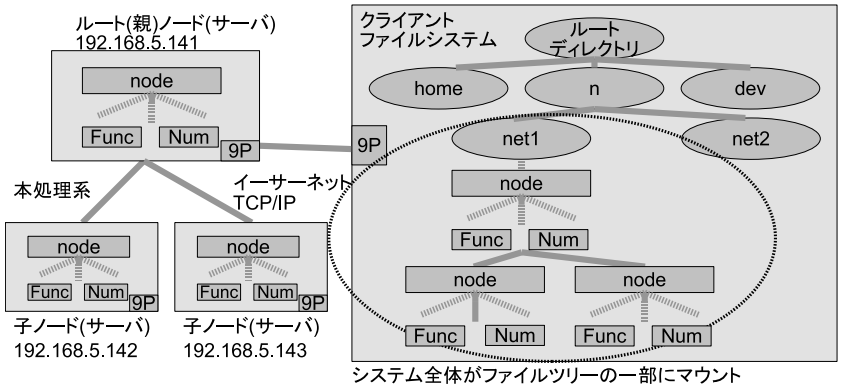

図 3 システムのプログラマから見たオブジェクト

等を行う必要がないことも本方式の特徴である．

7.3 ノードが提供する機能

クライアントが主に利用するノードの機能としては以下の物がある．

- VM のコードを実行し計算結果をファイル入出力の結果として出力する
- 9P プロトコルスタックを用いファイル入出力ベースのインタフェースを提供する

プログラマは利用者が利用する機能に加えて以下の機能を活用する．

- ノード上で動作するコードを VM にロードする
- ノードがクライアントとして他のノードを 9P プロトコルでマウントする
- プロトタイプチェーンなど VM 同士の連携が必要な処理を 9P プロトコルで実現する

これらの機能を利用し，ユーザはノード上のプログラムが仮想化されたファイルを読み込むことで，プログラムの実行を指示して結果を取得する．従来手法であれば，ユーザは手続きの呼び出しを指示し，値が返ってくるまで待つ命令を明記する必要が有ったが，ファイルの読み込み一つに集約できる．

また，プログラマはファイルに仮想化されたノードの制御用のファイルに値を読み書きすることで，コードの転送やノードの状態の確認を行う．アクセス透過性の高さを生かしノードの制御もネットワーク経由で行える．これも VM が特定のファイルに対する入出力を特殊な意味として解釈する本処理系の特徴である．更に，VM 同士の連携が必要な 7.6 節で後述するノードを超えたプロトタイプチェーンの機能もファイル入出力で行うことで，ノードがサポートすべき通信プロトコルを 9P プロトコルに集約することでリソースを節約できる．

7.4 ECMAScript のオブジェクトとファイルツリーの対応関係

9P プロトコルを用いて ECMAScript の各プロパティにアクセスするため，ECMAScript の各要素とファイルツリーで仮想化される対象をまとめたものが表 1 である．ECMAScript は名前空間が無いので，本システムではシステム全体を同一の名前空間とする．オブジェクトがノードの機能を記述する単位となり，ディレクトリに対応付ける．オブジェクトのメンバとなるメソッドや変数がファイルとして仮想化される対象になり，グローバル変数もファイルへ仮想化する．new 命令によってオブジェクトとしての実体を持つことがファイルツリーへの仮想化とする．

本システムではオブジェクトがディレクトリとして仮想化され，オブジェクトの継承関係はディレクトリ内にディレクトが存在する形で階層化する．ECMAScript なのでプロトタイプチェーンを用いて継承元のプロパティを活用できるが，これはディレクトリをルート方向に辿っていくことで実現する．これにより，本システムではシステム全体でルートノード

表 1 ECMAScript のオブジェクトとファイルツリーの対応関係

ECMAScript	ファイルツリー
オブジェクト	ディレクトリ
組込みオブジェクト	ディレクトリ
メソッド	ファイル
メンバ変数	ファイル
グローバル変数	ファイル
new	ファイルツリーへの仮想化
継承関係	階層関係
プロトタイプチェーン	ファイルツリーの遡り

のルートディレクトリを頂点としたファイルツリーを形成し，ノードを跨ぐプロトタイプチェーンも，9P プロトコルを利用した VM 同士のファイル入出力に変換される．

7.5 本システムでの ECMAScript のオブジェクト

ECMAScript でノードのコードを記述する際に，ノードの機能はオブジェクトを単位として記述する．そこで，オブジェクトが配置されるノードを指定したり，ディレクトリに仮想化される際のディレクトリ名を指定する特殊なプロパティを独自に設定する．プログラマが記述したソースコードをコンパイラが解析し，どのノードにどういった手続きを行わせ，通信の相手はどのノードである，と言った煩わしい設定を自動的に処理する．今回は TCP/IP での利用を前提としているのでオブジェクト毎に特殊な意味を持つ変数として，ip と name を追加する．ip はオブジェクトが設置される IP アドレスの値を格納する変数であり，name はファイルツリー内でオブジェクトがディレクトリに仮想化される際の名前となる．ECMAScript で記述する際に，オブジェクトの名前をディレクトリにするとオブジェクトの名前は重複できないので，同じ機能を持つオブジェクトが異なる名前となり分かり辛くなる．クラスの名前をディレクトリにすると，同じクラスをオブジェクトにして機能を後から追加したオブジェクトは同じノードに設置出来なくなってしまう．よって，オブジェクト毎にディレクトリの名前を設定できることが望ましい．上記の情報を元にファイルに仮想化するのに必要な情報をコンパイラが取得し VM の実行ファイルに埋め込む．

ユーザのアクセスコントロールのために，ユーザがアクセス可能なプロパティをプログラマが指定する識別子も指定する．独自に拡張したシンタックスとしてファイルに仮想化するメソッドや変数を明示化する識別子である visible を追加する．visible でファイル化されているメンバのファイル化を取りやめるための単項演算子として disappear も追加する．これらを組み合わせることで，動的にユーザに見せるファイルツリーを変更できる．

表 2 本処理系の ECMAScript のオブジェクト

オブジェクト	プロパティ
Array	concat，join，pop，push，reverse，shift，slice，splice，sort，unshift，length
Boolean	特に無し
Function	特に無し
Number	MAX_VALUE，MIN_VALUE，NaN， NEGATIVE_INFINITY，POSITIVE_INFINITY
Object	特に無し
String	length，append，charAt，charCodeAt，concat， ndexOf， fromCharCode，lastIndexOf，slice，split，substr，substring
追加オブジェクト	追加プロパティ
Object	ip，name，visible，disappear
Device	周辺装置はハードウェアの構成によって異なるのでプロパティも異なる

7.6 ノードのプログラミングモデル

ファイルへの仮想化を含み，他のノードの計算機資源を活用する疑似コードを示す．どちらの疑似コードも通信に関する記述は明記されていないように見えるが，他オブジェクトのプロパティの参照とプロトタイプチェーンによって実現できる．

ソース 1 に示すコードのように，システム全体が同一の名前空間内なので，同一ノードではないノードの計算機資源にも透過的にアクセスできる例である．実際は VM が 9P プロトコルを用いての対象となるノードの計算機資源を検索し，該当するプロパティが仮想化されたファイルを読み込むことに対応付けられる．

プロトタイプチェーンを利用してノードを超えた計算機資源の利用をするサンプルコードを示す．ソース 2 に示すコードは，オブジェクトをどのノードに配置するかのコードは含んでいない．TempSuper は親ノードに設置され，Temp は複数の子ノードに設置されることを想定したコードである．Temp では visible を用いることでファイルに仮想化する変数を明示している．Temp の comp_with_average で親ノードの Temp にプロトタイプチェーンでアクセスし，子ノードで取得した温度データを記録している．これは子ノードの情報を親ノードの提供するファイルツリー内の該当するファイルに値を書き込むことで対応する．プロトタイプチェーンは，子ノードが親ノードの提供するファイルツリー内をファイル入出力で探索し，見つかるまで親ノードを一つずつ辿りながら探索することで実現する．また，全子ノードの平均温度の計算は親ノード内で average として計算させ，計算結果だけを子ノードは利用している．プログラマは通信プロトコルの明記をせずに，ECMAScript のシンタックスでノード間の協調動作をコーディングできる高いアクセス透過性を示す例である．

ソース 1　ファイルへの仮想化を含むサンプルコード

```
/*ノードBのオブジェクト*/
function Speaker(ip, name) {
    this.ip = ip;
    this.name = name;
}
Speaker.sound = function() {
    beep();
}
/* ファイルツリーへ仮想化 */
node_B = new Speaker(ip_B, speaker_B);

/*ノードBの資源を活用したいのノードAのオブジェクト*/
function SpeakerClient(ip, name) {
    this.ip = ip;
    this.name = name;
}
SpeakerClient.visible.temp = function() {
    /* ファイルに仮想化する変数を指定するvisible */
    if (dev.ad1 * (変換式) > (危険温度)) {
        node_B.sound();
    }
}
/* ファイルツリーへ仮想化 */
node_A = new SpeakerClient(ip_A, client_A);
```

ソース 2　プロトタイプチェーンを利用するサンプルコード

```
/*ファイルに仮想化されない親ノードAのオブジェクト*/
function TempSuper() {}
TempSuper.prototype.temps = new Array[num_of_nodes];
TempSuper.prototype.average = function() {
    var temps_sum = 0;
    for (i=0; i<num_of_nodes; i++) {
        temps_sum += temps[i];
    }
    return temps_sum / num_of_nodes;
}

/*ファイルに仮想化される手続きを含む子ノードCのオブジェクト*/
function Temp() {}
Temp.prototype = new TempSuper();
/* ファイルに仮想化する変数を指定するvisible */
Temp.visible.temp_c = function () {
    return dev.ad1 * (変換式);
}
/* プロトタイプチェーンの利用例 */
Temp.visible.comp_with_average = function (my_number) {
    temps[my_number] = this.temp_c;
    if ( teis.temp_c > average() ) {
        beep();
    }
}
/*ノードとオブジェクトの設置場所の対応もECMAScriptで記述*/
```

7.7　VM が提供するファイルツリー

ソース 1 とソース 2 をファイルツリーに仮想化した例を図 4 に示す．左半分がソース 2 のファイルツリーで，子ノード C が提供するファイルツリーが親ノードのファイルツリーのディレクトリの中に取り込まれており，ノードを超えてもディレクトリの階層がオブジェクトの継承を表すのでアクセス透過性が高い．右半分がソース 1 のファイルツリーで，他のノード B にあるオブジェクトのプロパティを参照することをファイルツリーを介してアクセスすることで表している．プログラマが利用するファイルツリーは継承関係やノードの協調関係を確認するためのものなので，システム全体で一つのツリーを形成することが重要である．ルートディレクトリの下にはバイトコードから読み取った結果，ファイルツリーに仮想化されるオブジェクトがディレクトリとして存在する．これをオブジェクトディレクトリと呼び，オブジェクトディレクトリ以下は ECMAScript のソースコードからコンパイラと VM が自動的に生成したファイルツリーである．プログラマがツリーの形状に関して行う記述はファイルへの仮想化の指定のみであり，クライアントがユーザのときのみ提供されるファイルツリーに反映される．各オブジェクトディレクトリの下にはファイルに仮想化される指定を受けた変数がファイルとして存在する．このファイルに読み書きを行うことで計算結果を取得したり変数に値を格納したりする．オブジェクトディレクトリの下にディレクトリが存在する場合，オブジェクトの継承関係を意味しており ECMAScript の場合プロト

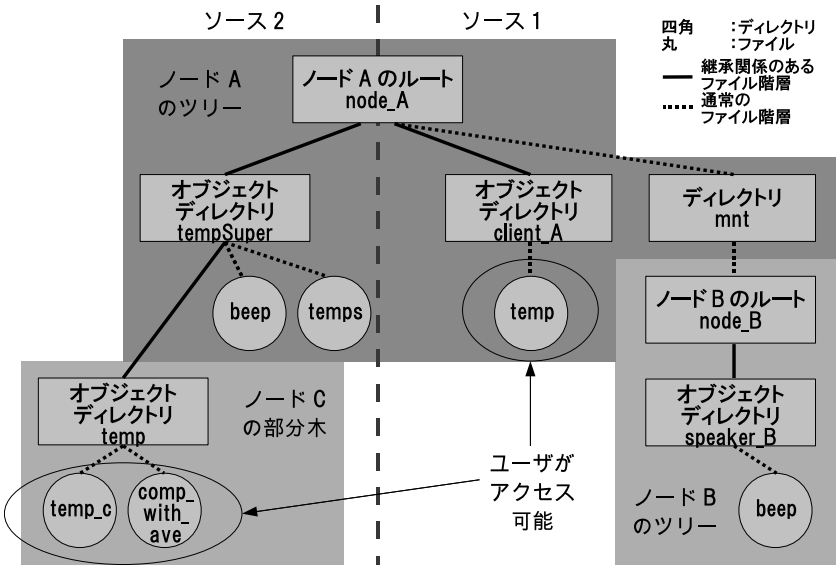

図 4 VM 提供ファイルツリー例

タイプチェーンの対象となる．

他にもルートディレクトリの下には dev ディレクトリ，mnt ディレクトリ，ctrl ディレクトリが存在する．これらのディレクトリやファイルはノード毎に存在し，それぞれ特殊な意味を持つ．dev ディレクトリはノードの周辺機器がファイルに仮想化されているディレクトリで，ECMAScript で記述するソースコード中では組込みオブジェクトとして用意されている AD コンバータなどに対応する．他のノードの周辺機器もファイル入出力によって対応付けらた他オブジェクトのプロパティ参照で柔軟に活用できる．dev 内の各オブジェクトは周辺機器にアクセスする為のメソッドや設定用の変数を保有している．mnt ディレクトリはノードが他のノードの計算機資源を利用する際にマウントするディレクトリである．ctrl ディレクトリはノードの制御・ノード間連携用の機能を実現するファイルを提供する．com ファイルを通してノードの制御を行い，code_*ファイルは実行ファイルを書き込む先としてクライアントや他の VM に生成された実行ファイルが書き込まれる．table ファイルはノードの継承元の IP 等が記録されておりプロトタイプチェーンで活用する．

8. 実装と評価

本研究では，フラッシュROM を 256KB，RAM を 32KB 搭載した 32bitCPU の ColdFire MCF52233 をターゲットに実装した．試作で作成したシステムとしては，ノードの AD コンバータにサーミスタを接続したものを 3 個用意し，Linux からサーミスタの温度情報を取得するものがある．VM で行うユーザコードの処理としては，AD コンバータの値を取得し温度情報に変換することである．サーミスタの抵抗値と温度から AD コンバータで取得した値を温度に直す為，幾つかの条件文で線形近似する区分を切り分け，四則演算で温度を算出した．対応した 9P プロトコルとしては，認証に用いる auth メッセージ以外は VM 上で解釈できたので，Linux で用いるコマンドの mount，ls，cd，cat，echo に対応する 9P プロトコルを受け取り，VM の処理を分岐させた．クライアントとしての Linux から発行される 9P プロトコルに VM が応答し，温度情報はファイルの出力として取得される，ノードで過去の温度情報を保持しておく処理は行わない．Linux では cat コマンドを利用して 9P プロトコルの発行を行え，shell スクリプトでノード間の平均温度を計算したり，web サーバを利用してノードの温度変化を一定時間毎にグラフにすることを行った．ノードの利用において 9P プロトコルを直接発行するコマンドは利用しておらず，ファイル入出力のみで実現できた．

メモリ使用量に関して，VM の必要とする ROM サイズは 9P プロトコルスタックの一部を含めて 200KB 程度であり，使用 RAM サイズは VM と 9P プロトコルスタックの一部を含めて 30KB 程度となっており，組込みの 32bitCPU には十分である．

VM の処理性能は同じマイコン上に実装されている SilentC[6] インタプリタとネイティブ

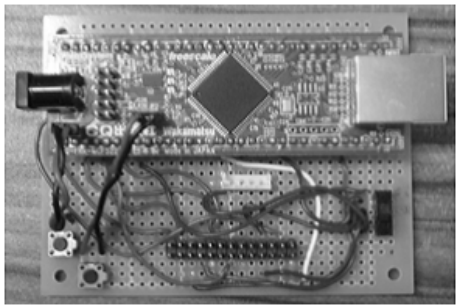

図 5 MCF52233 基板

表 3 MCF52233 上での実行時間

	本処理系	SilentC	ネイティブ
3 重ループ	4.6[ms]	1.19[s]	0.13[ms]
再帰呼び出し	2.2[ms]	0.29[s]	0.08[ms]
和演算	0.6[ms]	1.25[ms]	0.2[μs]

コードと動作性能を比較し，表 3 に示す．3 重ループは各要素 10 の中でインクリメントを行い，再帰呼び出しは 10 までのフィボナッチ数を求め，和演算は 1〜10 の和を計算した．VM の実行時間は VM のスタック領域を確保するところから実行ファイルをロードしてコードを実行し，スタックを開放するまでの一連の時間で計測している．処理時間として，本研究の方式はネイティブコードに比べ 30 倍近い実行時間がかかるが，インタプリタである SilentC の 200 分の 1 程度の実行時間で処理できる．

9. 考　　察

本システムにおいてノードに求める主な処理内容は，ノードに搭載されている周辺装置のデータを取得して簡単な計算などの処理を施し，ユーザが利用しやすいデータに変換することだと考えられる．これはセンサーネットワークのような，時間制約が緩い分散システムを想定しており，主な計算やノードから取得したデータはユーザ側で管理することで，ノードの処理速度とのバランスを図る．よって，ノードで行う処理が記述された実行ファイルのサイズも小さく収まるので，少量の本実装でも十分な計算機資源が確保されていると言える．ユーザはシステム中に大量に存在するノードから効率良くデータを集めることが求められるが，これはローカルファイルツリーにマウントされたノードに対してワイルドカードを用いながらアクセス透過性の高い本システムのファイル入出力だからこそ実現できることである．

実装面では，ECMAScript のファイルへの仮想化情報など書き換えが生じないデータを ROM に書き込むことで，RAM の使用量は現在よりも削減できる．9P プロトコルのクライアントとしての機能を追加するリソースは残っている．ROM に関しては 50KB 程度の余裕が有り，ノードの書き換えをノード自身が行えるので，実行ファイルを置くのに利用できる．実行ファイルのサイズとしては，フィボナッチ数列を求めるもので 130 バイト程度なので十分に余裕が有る．

表 3 から，ネットワークのレイテンシーを考えると ms オーダーで処理を終わらせることができる本処理系は，ネットワークを利用する分散システムの処理系として十分な実行性能を有すると言える．本処理系では VM の計算領域としてのスタックメモリの確保に 5ms，スタックメモリの解放に 2.5ms と計測時間において大きな割合を占めている．これらはノードの初期化に用いるだけなので，ノードがシステムとして動作している間の実行時間には含まれず，表 3 には含んでいない．また，VM はコードを実行する前に組込み関数や参照テーブルを構築する．これらは，機能拡張と共に増加すると思われるが，現時点で 0.2ms 程度なので問題とならない．

10. ま と め

本研究では分散システムに ECMAScript と 9P プロトコルを用い，ノードをオブジェクトと捉えてプログラマがシステム全体を透過的にコーディングできる，アクセス透過性の高い分散システムの構築法を提案した．

今後の課題は，9P プロトコルの実装を VM 上で進めクライアント側の 9P プロトコルの発行に対応し，プロトコルに応じた処理を実装することである．その後は VM 間の連携機構を実装していくことである．

参 考 文 献

1) Common Object Request Broker Architecture: Core Specification: Object Management Group, http://www.omg.org/cgi-bin/doc?formal/04-03-01.pdf
2) Arduino Software: www.arduino.cc/
3) vita nuova: Inferno, www.vitanuova.com/inferno/
4) Bell-labs: Plan 9 from Bell Labs Fourth Edition , plan9.bell-labs.com/plan9/
5) Bell-labs: Plan 9 File Protocol, 9P , plan9.bell-labs.com/sys/man/5/INDEX.html, 2008.
6) SilentSystem: OS1 プログラミングマニュアル, www.silentsystem.jp/download/OS-1_Programing_06.pdf (2007).
7) Yoshimasa Niwa, Satoru Tokuhisa, Masa Inakage:Talktic: a development environment for pervasive computing applications , ACE '08 Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology
8) Q. Cao, T. Abdelzaher, J. Stankovic, K. Whitehouse, L. Luo : Declarative Tracepoints: A Programmable and Application Independent Debugging System for Wireless Sensor Networks, In Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (ACM/IEEE SenSys), 2008.
9) Philip Levis, David Culler: Mate: a tiny virtual machine for sensor networks ACM SIGOPS Operating Systems Review Volume 36 pp.85-95, 2002.