# Plan9を用いた分散組込みシステムのオブジェクト指向プログラミングシステムの提案

盛合智紀[†] 並木美太郎[††]

本研究では，分散システムのノードとして言語処理系の VM(仮想機械) を搭載した組込みシステムを用い，分散透明性を有した分散システムを構築する手法を提案する．分散システムの通信プロトコルに TCP/IP 上で，Bell 研究所が開発した Plan9 と，ネットワークファイルプロトコルである 9P を用いる事で，ネットワークを意識せずに，ノードの手続きをファイルに仮想化し，ファイル入出力のように扱うことで高い透過性とアクセスの統一性を実現する．本発表では，上記のシステムの試作として，オブジェクト指向言語ではない，言語 C のサブセット C -- ' のコンパイラとその VM を試作した．その結果，VM の ROM サイズは 80KB，RAM サイズは 15KB 程度となり，実行時間については，大体ネイティブコードの 400 倍弱，同ハードウェア上で動作するインタプリタの SilentC と比べて 1/10 程度となり，ノードのプログラムを実行するために十分な実行時間であることがわかった．本稿では，上記の 9P ベースの分散システムの考え方，試作した結果について報告するとともに，オブジェクト指向言語をこれらのノードに 9P 向けに実装する方式について述べる．

# Object-oriented Programming System for Distributed Embedded System using Plan9

Tomoki Moriai[†] and Mitaro Namiki[††]

This paper describes programming system for distributed embedded system with location transparency using TCP/IP networkd and 9P protocol which is desined for Plan9 operating system. This system supports Remote Procedure Call interface as file I/O interface of Plan9. A prototype system includes embedded server implemented with Virtual Machine S91 for C-- ' subset language of C. As the result of prototyping, using size ROM size is about 25KB, RAM size is about 15KB. In the compared execution time of native code and SilentC interpreter, S91VM marked about 375 times slower than native code and S91VM marked about 10 faster than SilnetC. This paper reports the result of the prototype using 9P protocol and describes the method using 9P for object-oriented language.

## 1. はじめに

本研究では，分散システムにおける組込み機器の資源を，Plan9[1] 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 プロトコル[2] を用いて透過的に管理する，分散システムの構築法の提案と，提案手法のための，センサーノードのような少ない資源の計算機環境下でも動く言語処理系の VM(Virttual Machine) と，言語処理系の設計と試作を行った．

† 東京農工大学大学院情報工学専攻
Department of Computer and Information Sciences, Tokyo University of Agriculture and Technology
†† 東京農工大学大学院共生科学技術研究院
Graduate school of Engineering, Tokyo University of Agriculture and Technology

### 1.1 背　景

近年，光ファイバーや ADSL といった物に代表される情報インフラの拡充によって，いつでもどこ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る。その結果として，小さな組込み機器の上で，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，どこにでもある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．

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．

### 1.2 問題提起

組込み機器を利用した分散システムを構築する際に，次の二つの課題が残る．一つ目が，分散システムの資源管理に関してである．二つ目が，センサーノードで行う処理を記述する際の，組込みプログラム開発に関する問題である．

**(1) 分散システムの資源管理**

古典的な分散システムでは，OSI 参照モデルで言う所の 4 層目に当たるトランスポート層の通信プロトコルである TCP や UDP をコード中で明示的に利用し，ネットワーク通信の利用を意識したプログラミングを求めるため，分散システムの導入の敷居を高めていた．最近では，ミドルウェアを利用し，分散システムを構築する際の通信プロトコルスタックとして OSI 参照モデルの 5 層以上に当たる IIOP[3] 等のプロトコルを提供し，プログラマに細かな通信内容を意識しない環境が提供され始めた．

しかし，ミドルウェアを利用するには，ミドルウェアの機能を提供するライブラリが開発環境に応じて存在する必要がある．また，ミドルウェアを利用してもネットワークを意識したコネクションの確立や，データの送受信待ちは明示的に示す必要がある．これは，プログラマにネットワークを意識させることとなり，分散システムを構築する際で重要な要素である，ユーザーにリソースの存在場所を意識させない分散透過性の視点から見ると，改良の余地がある．

CORBA を利用し，スタブやスケルトンを作成して実現する場合，IDL によるインタフェースの記述が必要不可欠であり，IDL を元にクライアントプログラムを記述する必要がある．クライアントプログラムはサーバプログラムの設計内容を意識する必要が生じる．更に，省資源しか有さない計算機に，CORBA の提供するサービスをサポートさせるには，CORBA の機能が豊富すぎるという見方もできる．

**(2) 組込み機器の開発**

組込み機器に対するシステム開発では，各ベンダーが開発した各機器やサーバのアーキテクチャに応じた開発環境を揃える必要があり，アーキテクチャの種類が豊富にある分，様々な開発環境も求められることとなる．アーキテクチャの違いに対応するには，アーキテクチャ毎にソースコードを変更する必要があり，変更したソースコードを組込み機器に焼き付ける必要がある．これは，物理的に離れた位置に点在する分散システムを考えたときに，個々のアーキテクチャに応じたプログラミングを行い，ROM に格納するなど手間がかかる．

## 2. 先行研究

センサーネットワークないしは，分散システムに関する先行研究を示す．

**(1) LiteOS[4]**

センサーネットワークを構築するノードを，ユーザーから見て，ファイルツリーの形式で管理できることが特徴である．管理のために，LiteShell と呼ばれる専用のツールを使うことで，UNIX ライクなコマンドによる操作を用い，ファイルツリーを移動して，ノードの資源を管理できるので位置透過性は高い．MicaZ や IRIS[5] といった，センサーネットワークを構築する際に多く用いられるハードウェアを対象とした OS である．センサーネットワークを利用するには，サポートされたハードウェアを PC に接続し，接続したハードウェアとノードにそれぞれ専用の LiteOS を載せる必要がある．LiteShell を通して利用する必要があるので，ノード資源を利用できるのは一部の Java 等が用意されている環境に限定されるので，アクセス透過性は低い．

**(2) Inferno[6]**

Bell 研究所が Plan9 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られる。また、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．全ての資源がファイルに仮想化されることは Plan9 同様なので，位置透過性もアクセス透過性も高いといえる．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 は必要になるので，今回の様な省資源の計算機を対象とする場合，利用することは困難である．

この様に，省資源しか有さない計算機を対象とする場合，従来の方法では，位置透過性を確保するために専用のツールを利用し，アクセス透過性の低下を招く

か，ある程度の計算機資源を要求することになってしまう．

## 3. 目　　的

本研究では分散システムの通信プロトコルにTCP/IP を利用した 9P プロトコルを用いることで，ノードの手続きをローカル空間でのファイル入出力のように扱う，位置透過性，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．これにより，ソケットの生成など，ネットワークを意識する必要が無くなり，RPC(Remote Procedure Call) の実行をファイル入出力のみで行える．加えて，ノード間の継承を利用することで，ノードの処理内容を一つずつ記述する必要性を排除し，ノードのソフトウェアを開発するプログラマにとっては開発コストを減らせられる．また，提案した分散システムを実現するために，数 KB 程度の省資源の計算機でも動作するスタックマシン型の VM と，ECMAScript のようなオブジェクト指向言語を用いた処理系を提供することを目的とする．

この方式により，クライアントに当たる PC は，9P プロトコルさえ利用できれば，他に必要なツール等は無く，本研究によって提案するノードを利用できることを意味しており，ノード上の手続きをファイルに仮想化することで，ファイルの読み込みが RPC に該当することが，本方式の特徴である．つまり，クライアント/サーバ間の通信のみならず，資源管理においてファイル入出力を用いる単一的なモデルを提供できる．

さらに，VM を利用することで，アーキテクチャ依存部分以外のコードを使いまわすことがし易い．また，ノード上の手続きをネットワーク越しに変更することも容易になる．物理的に離れた所に点在しうる，分散システムの利用を考えた際に，有用な機能である．

また，ノード間のプログラムの継承関係を持たせることを可能にし，ノードのプログラム内容を変更することで，変更が加わったノードを継承するノードの処理内容にも変更が反映させる．これによって，個々のノードに個別対応することなく，全体のノードを一括して管理できることを目指す．

## 4. 試　　作

本試作では，主にノードの手続きをファイルに仮想化し，ノードの利用者に対して透過的なファイルアクセスでノードの資源を活用できることを主目的とした．

### 4.1 設 計 方 針

以下において，クライアントは RPC を呼び出す側を表す．一方で，サーバはクライアントからの RPC の要求を処理する側で，サーバは本研究において作成される独自の処理系を載せたノードを指す．本研究では，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 プロトコルを理解するための機構である．9P プロトコルを処理できれば，PC がクライアントである必要は無く，クライアント側に手を加えずに，サーバの資源を利用できることが特徴である．

また，サーバ上で動作させる S91 と呼ばれるスタックマシン型の VM を作成し利用する．この S91VM 用の実行コードを出力するコンパイラとして，C 言語ライクなシンタックスを持つ C -- ’ 言語という物を用いる．C -- ’ は C 言語のサブセットに，S91VM において RPC の対象としてファイルに仮想化してクライアントに見せる手続きを選別するための修飾子等を言語仕様に追加した仕様となっている．S91VM がサーバ上では 9P プロトコルスタックを保持しており，通信内容を判断して適切な処理を行う．S91VM ではコンパイラから得た情報をもとにファイルに仮想化する手続きを特定し，ファイルツリーを自動的に生成する．ファイルツリーは 9P プロトコルを通してアクセスされ，手続きが仮想化されたファイルに対するアクセスが発生すると S91VM がファイルに対応する手続きを実行し，実行結果をファイルの読み込み結果として返す．S91VM はノード上の手続きをファイルに仮想化することと，9P プロトコルと言語仕様に直接な対応付けを行う．

### 4.2 システムの構成図

システムの全体図を示す．図 1 では簡略化のため，サーバの保持するファイルツリーを省略している．サーバの保持するファイルツリーの構造については，5 章で述べる．

図 1 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，プログラム中の名前に対応しサーバ上の VM で動作する命令コードから生成される．VM の実行を管理する名前や，RPC の結果を取得する，オブジェクトの階層になっている．そこで，上記のファイルとして仮想化す

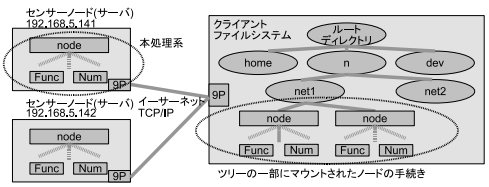


**図 1**　システム構成図

表 1　C -- ' 用 API の設計

| 名前 | 機能 |
|---|---|
| mount(ip,*dir_fp) | 他のノードを mount する |
| umount(*fp) | 他のノードを umount する |
| fopen(*fname,type) | 通信用ファイルを開く |
| fclose(*fp) | 通信用ファイルを閉じる |
| fgetc(*buf) | 通信先から値を取得する |
| fputc(c) | 通信先に値を送る |

るための情報として，関数名や変数名が別途必要になるので，サーバで動作させるコードファイルに，必要な情報を埋め込むことで，サーバ上で関数名の設定等の処理を行う必要がないことも，本方式の特徴である．

### 4.3　ノードの提供する機能

ノードの提供する機能としては以下の物がある．

- ノード上で動作するコードを，クライアントから取得し，ロードする
- ロードしたコードを管理する為のインターフェースをファイルに仮想化して，9P プロトコルを利用してクライアントに提供する
- 複数のコードを同時に実行する

ノードはクライアントからの RPC の要求を受け取り，内容に応じた処理の結果をクライアントに送り返す．RPC を実行する際のインターフェースにファイル入出力を用いており，クライアントはノード上の RPC の対象となる手続きを仮想化したファイルを読み込むことで，RPC の実行の指示と結果の取得を同時にできる．従来手法であれば，RPC を呼び出し，値が返ってくるまで待つ命令を書いたが，ファイルの読み込み命令一つに集約される．

ノードの提供する機能を利用するために，ノード上で操作するコードをロードする方法と，クライアントからノード内の資源を利用する方法について述べる．

### 4.4　ノードの処理内容の記述方法

ノード内の処理を記述するには，C -- ' 言語を用いてコーディングを行う．C -- ' によって記されたコードはコンパイラによって S91 バイトコードにコンパイルされる．C -- ' では特定の識別子が付いている関数をファイルに仮想化するので，関数の形で RPC の対象となる手続きを記述する．関数の戻り値がファイルの内容として読みだされる．

図 2 に図 3 でファイルツリーに仮想化をするサンプルコードを示す．

サーバ上の C -- ' プログラムにおける，クライアントや他ノードとの通信用のファイルを利用する API を示す．主に図 3 における，connection ディレクトリ内の通信用ファイルを利用するための API である．

GetTemp.c--

```
int a;                  /* グローバル変数 */
int final b = 1;        /* 書き換え不可能 */

setrpc int Get() {  /* RPC対象関数 */
      int temp;

      /* sence_adの呼び出し */
      temp = sence_ad();

      /* RPCの結果として返す */
      return temp;
}

int sence_ad() {   /* RPC対象外関数 */
      ・・・
}

/* 引数付きRPC対象関数 */
setrpc int GetArg(int num) {
      ・・・
}

void main() {
      while(1){・・・}/* RPC実行時以外の処理 */
}
```

図 2　C -- ' で記述されたノードのソースコードの例

### 4.5　クライアントによるノード資源の利用方法

クライアントからノードの資源を利用するには，クライアントから 9P プロトコルを利用して，ノードをクライアントの持つファイルツリーの中に mount する．ノードの提供するファイルツリーを部分木として，クライアントの持つファイルツリーの一部に張り付ける．クライアントのプログラムからノードの提供するファイルを利用するときは，クライアントのファイルツリーの一部に取り込まれているため，プログラムからはローカル空間のファイルにアクセスする方法と全く同じアクセス方法でノードを利用できる．ノードのファイルにアクセスがあった場合は，クライアントのファイルシステムがその旨を解釈し，通信に関しては 9P プロトコルスタックに帰着する．

mount されたノードの資源を利用するには，4.4 に記したノードが RPC 対象手続きを仮想化して提供するファイルをインターフェースに用いる．ノード上のプログラムの実行を指示し結果を取得するには，クライアントのファイルツリーの一部に取り込まれている RPC 対象手続きに該当するファイルを読み込めば良い．

## 5.　VM を用いた資源管理方法

### 5.1　VM の提供する資源とファイル木による仮想化

クライアントのファイル空間に取り込まれる際の

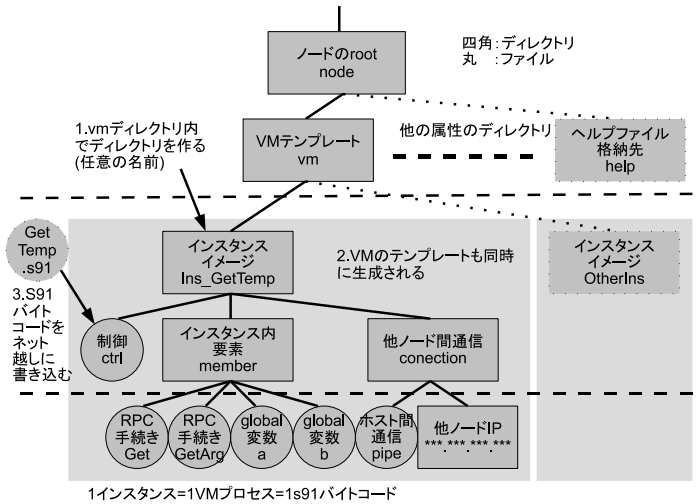


**図 3** VM 提供ファイルツリー例

VM が提供する資源のイメージと役割を述べる．ノードが提供するファイル木がクライアントのファイル木の一部となり，クライアントのプログラムからはローカル空間のファイル入出力と全く差異が無くなることが大きなメリットである．ノード上のプログラムの名前を元に C -- ' コンパイラと S91VM が連携して，自動的に RPC の対象となる手続きやグローバル変数を同じ名前のファイルにし仮想化する．そのため，ノードのプログラマーはインタフェースとなるファイルへの仮想化に関しては考慮する必要は一切無い．今回の設計では，1 インスタンスが 1 つの VM になるように設計を考える．VM 内で実行される単位は，一つのロードされた S91 バイトコードである．

全体の構造としては，ルートディレクトリがあり，その下に VM 上で実行する RPC のインスタンスのテンプレートである vm ディレクトリが存在する．他のディレクトリも同じ階層に同時に持つこともできるが，例として，図 3 では，help ディレクトリも IP アドレスディレクトリの下に位置する．

### 5.2 インスタンスの提供するインターフェース

図 3 の VM インスタンスである Ins_GetTemp ディレクトリ以下に含まれるファイルやディレクトリの機能を説明する．VM インスタンスである GetTemp ディレクトリ以下には，インスタンス内の要素を格納する member ディレクトリ，他ノードやホストと通信を行うためのファイルを格納した connection ディレクトリ，インスタンスを実行する VM 内でのインスタンスの状態を管理する ctrl ファイルが存在する．これらのディレクトリやファイルは S91 のバイトコードをロードすることでバイトコードから自動的に生成される．図 3 のように，最初からツリーが構成されている訳ではない．ルートディレクトリと vm ディレクトリのみが最初は存在する．この状態は，ノードに S91 バイトコードがロードされていない．vm ディレクトリが vm にロードされる手続きのテンプレートを意味し，vm ディレクト内 (図 3 の VM テンプレート) に新規にフォルダを作成することが，テンプレートの継承に該当する．継承された個別のテンプレート当たるフォルダには，手続きを管理するためのインターフェースが，ファイルやディレクトリの形で VM によって用意されており，それらのファイルを利用して，S91 バイトコードを書き込み，インスタンス化する．生成されたインスタンスが RPC のインターフェースとなるファイルを保持しており，このファイルに対するファイル入出力が RPC の実行となる．

member ディレクトリは，S91 バイトコード内で定義されている，RPC の対象となる関数と，グローバル変数をクライアントから参照するためのディレクトリである．member ディレクトリ直下の各ファイルに Write や Read を行うことで，RPC の対象となる関数に引数を与えたり，RPC を実行し，その戻り値を取得できる他，変数にもアクセスできる．

conection ディレクトリは，ノードを mount しているクライアントとの通信を行う為のファイルで，Write と Read を行うことでデータの授受を行う．また，ロードした S91 バイトコード内で，mount した他ノード

が配置される先も connection ディレクトリである．

ctrl ファイルは，インスタンスが実行される VM の管理を行うためのインターフェースとなるファイルである．詳しくは 5 章で後述する．図 3 に関して，ノードの mount，S91 バイトコードのロード，インスタンスの初期化，RPC の実行，インスタンスの消去と資源の解放の流れを Unix のコマンドを利用した方法で記す．端末からのコマンドのみで RPC を実行できることが特徴である．これは，Unix のコマンドが 9P プロトコルに翻訳された結果であり，9P プロトコルに対応付けができれば，クライアントによらず同様の操作内容で RPC を実行することができる．例を図 4 に示す．

図 4 に示されるように，ctrl ファイルへの特定文字列の書き込みによって，S91VM を初期化する．ctrl は他にも，ノードのプログラムを上書きするモードの指定や，プログラムの消去にも用いる．

S91 バイトコードは C -- ' 言語を利用して記述されたノードの処理内容をもとにコンパイラと VM によって自動的に生成され，この中にファイルの名前などの必要な情報が含まれる．C--' は C 言語のサブセットではあるが，独自に拡張したシンタックスとして，RPC の対象とする関数と，それ以外の関数の扱いを区別する為の識別子である setrpc がある．setrpc が記述されている手続きは，コンパイラによって RPC の対象となる手続きと解釈される．RPC の対象となる手続きは，クライアントからノードを見るときに，ファイルに仮想化され，ファイル名等の必要な情報が S91 バイトコードに埋め込まれる．また，RPC の実行依頼を処理している時間外で動作する main 関数も存在する．S91 バイトコード内で RPC 対象の手続きとグローバル変数は，クライアントから利用される為のインターフェースとしてファイルに仮想化され，member ディレクトリに格納されている．

```
Linux(Plan9)

modprobe 9p

～ノードのマウント～
mount -t 9p -o proto=tcp
192.168.5.141 /n/node/

～VMテンプレートの継承～
mkdir /n/node/192.168.5.141/vm/Ins_GetTemp

～インスタンス化～
echo new > /n/node/192.168.5.141/vm/Ins_GetTemp/ctrl
cat GetTemp.s91 > /n/node/192.168.5.141/vm/Ins_GetTemp/ctrl

～インスタンスの初期化～
echo start > /n/node/192.168.5.141/vm/Ins_GetTemp/ctrl

～RPCの実行～
cat /n/node/192.168.5.141/vm/Ins_GetTemp/member/Get

～引数を与える～
echo 2 > /n/node/192.168.5.141/vm/Ins_GetTemp/member/GetArg

～RPCの実行2～
cat /n/node/192.168.5.141/vm/Ins_GetTemp/member/GetArg

～インスタンスの消去、資源の解放～
echo kill > /n/node/192.168.5.141/vm/Ins_GetTemp/ctrl
```

**図 4** ノードへのアクセス例

## 6. 9P プロトコルと VM 操作の対応

サーバ上で動作する S91VM における 9P プロトコルスタックは，通常の 9P プロトコルと異なり，操作対象のファイルやディレクトリによって RPC の実行指示や，VM インスタンスの状態を取得，などの特殊な操作を判断する必要がある．プロトコル自体は通常の 9P プロトコルを利用しているので，クライアント側の環境は一般的なものを利用できるが，サーバ側では送られてきた 9P プロトコルのメッセージに対して適宜異なる意味解釈を自動的に行っていることが特徴である．

そこで，VM インスタンスが提供するファイルやディレクトリに対する 9P プロトコルのメッセージと，特殊な操作が必要なメッセージに対するメッセージの解釈内容を対応付ける．表 2 に 9P プロトコルのクライアントからサーバに送信される T メッセージを列挙する．内容は，T メッセージの名前と含まれるパラメータである．[] 内に記されるバイト数で各名前のデータを示す．バイト数が文字の物は可変長であることを表す．tag はメッセージを識別する為の固有の数字であり，fid はファイルディスクリプタのようなもので，fid に対してファイルに関連付けられる．ただし，複数の fid が同一ファイルを指し示すことはあるが，一つの fid が複数のファイルを差すことは無い．

- Topen

fid で指定されたファイル/ディレクトリのアクセス権の確認と I/O の準備．mode はファイルを開いた後に認める動作を示し，読み，書き，実行，に加え，ファイルの切り詰めや，ファイルを閉じるときにファイルを消すことも指示できる．ファイルの入出力を行うときに実行されるので，Topen メッセージを受けとった時点で，S91VM が操作対象に特殊な解釈をする必要があるか判断する．

- Tcreate

fid で指定されたディレクトリに name で指定された名前のファイル/ディレクトリの作成を行う．その際に perm で指定されたパーミッションを，fid のパーミッションを考慮した上で割りつける．mode によって，ファイルの生成かディレクトリの生成かを判断する．新しくファイルが作られた場合，同時に mode で指定

表 2　9P プロトコル

| T-mesages | パラメータ |
|---|---|
| Tversion | tag[2]msize[4]version[n] |
| Tauth | tag[2]afid[4]uname[s]aname[s] |
| Tflush | tag[2]oldtag[2] |
| Tattach | tag[2]fid[4]afid[4]uname[s]aname[s] |
| Twalk | tag[2]fid[4]newfid[4]nwname[2]nwname*(wname[s]) |
| Topen | tag[2]fid[4]mode[1] |
| Tcreate | tag[2]fid[4]name[s]perm[4]mode[1] |
| Tread | tag[2]fid[4]offset[8]count[4] |
| Twrite | tag[2]fid[4]offset[8]count[4]data[count] |
| Tclunk | tag[2]fid[4] |
| Tremove | tag[2]fid[4] |
| Tstat | tag[2]fid[4] |
| Twstat | tag[2]fid[4]stat[n] |

された内容でオープンも行われる．その際に Tcreate 内の fid の値が，新しいファイルの fid として割りつけられる．Tcreate メッセージから実行される親ディレクトリの fid が分かるので，fid を利用して Topen メッセージ同様，特殊な操作が必要か判断を行う．

- Tread

fid で指定されたファイル/ディレクトリのデータ又は所属ファイルの属性を読み取る．fid から特殊な操作が必要なファイル/ディレクトリを対象としているか判断できるため，それに応じた結果を返す．

- Twrite

fid で指定されたファイルにデータを書き込む．fid から特殊な操作が必要なファイル/ディレクトリを対象としているか判断できるため，それに応じた処理を行う．

つまり，Topen メッセージと Tcreate メッセージを受信した際に，VM が特殊な解釈をする必要があるかの判断を行う．判断結果は，ファイルやディレクトリを識別するための fid と対応付けることができる．Tread メッセージや Twrite メッセージは，fid に割りつけられている特殊な解釈が必要かを判断し，求められる操作を行う．

connection ディレクトリでのディレクトリの生成と，member ディレクトリでのディレクトリの作成は，それぞれの親ディレクトリの特徴を継承するものとする．つまり，member ディレクトリ内に submember ディレクトリを作成したとき，submember ディレクトリ内でファイルを生成しても，member ディレクトリ内でファイルを生成した時と同様に，インスタンス内に新たなメソッドが追加されたとみなす．それ以外は，本処理系で特殊な解釈は行わず，通常の 9P プロトコルの持つ意味として，ファイルないしはディレクトリの生成を行う．

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 | 0.12[s] | 1.19[s] | 0.13[ms] |
| 再帰呼び出し | 0.03[s] | 0.29[s] | 0.08[ms] |
| 和演算 | 0.18[ms] | 1.25[ms] | 0.2[$\mu$s] |

## 7. 実　　装

今回は ColdFire MCF52233 をターゲットに実装を行った．同じマイコン上に実装されている SilentC[7] インタプリタとネイティブコードと動作性能を比較する．

また，メモリ使用量に関して，S91 の必要とする ROM サイズは 25KB 程度であり，使用 RAM サイズは S91 が 15KB 程度となっており，組込みの CPU では比較的メモリが多い．

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．全体として，本研究の方式はネイティブコードに比べ 1000 倍近い実行時間がかかるが，インタプリタである SilentC の 10 分の 1 程度の実行時間で処理できる．

また，サーバ上で動作する S91VM の 9P プロトコルスタックを一部を実装し，動作確認を行った．具体的には，サーバでは予め VM に登録されているファイル情報を用い，クライアントにあたる Linux の mount コマンドや，ls コマンド，cd コマンドに対する応答を実現した．cat コマンドを利用し，予め VM に載せてある RPC 対象となるコードを実行し，結果を文字列として取得する事も実現した．

## 8. オブジェクト指向の特徴を扱ったノードの提案

本設計では試作で行った目的に加えて，ノードの

プログラミングを行い分散システムを管理する際に，ノード間の継承を利用することで多くのノードを一元的に管理することを目的とした．

### 8.1 設計方針

継承の概念を利用しノード間での継承を行うことで，継承元の内容を継承先に引き継けることが特徴である．ここでいう継承とは，ノード上のプログラムを継承することである．継承先のノードは継承元のノードのプログラム変更に対して暗黙のうちに影響を受ける．多重継承を用いることで，複数の機能を個別の基幹ノードで管理し，複数の基幹ノードを多重継承するノードは基幹ノードの持つ機能を同時に保持する事が可能となる．基幹ノードへのプログラム変更によって，基幹ノードを継承する全てのノードの処理内容が変更されることがシステム構築上の特徴となる．

基本的な構成として，ノード上に S91VM が動作し，9P プロトコルを利用することは試作と変わらない．S91VM 用の実行コードを出力するコンパイラとして，ECMAScript をベースとした言語を利用する．ファイルに仮想化してクライアントに見せる手続きを選別するための修飾子等を追加した仕様となっている．

ノードの継承は mount に該当し，継承元のノードが継承先のノードを mount することでノード間の継承関係を構築することが特徴である．これにより，ノード一つ一つに継承元を知らせることなく，継承元の管理で一元的に継承関係を管理できる．

ノードの利用方法は 2 通り考えられ，一つはクライアントから 9P プロトコルを通して通常のファイルとして利用する方法であり，もうひとつは S91VM からノードを利用する方法である．後者のノードの継承として利用している場合，単にノードを mount して利用する場合と同様にファイル入出力を用いる．これは ECMAScript には本来存在しない仕様なので，ファイル入出力用の組込みクラスを定義して利用する．

### 8.2 ノードのプログラマに提供する機能

ノードはプログラマからプログラムを受け取り，プログラムに応じたファイルツリーを構築する．このとき，ノードの mount が特定ディレクトリ内に対して発生すると，相手ノードが本処理系を載せている場合はノードの継承と解釈する．継承によって，継承先のノードから継承元のクラスを利用できるようになる．

ノードの提供する機能としては以下の物がある．

- mount が発生し，mount 先が本処理系を載せている場合はノードの継承を行う
- ノード上で動作するコードを，プログラマから取得し，クラスを把握する
- ノードの周辺機器を利用するためのファイルに仮想化されたインタフェースを提供する

### 8.3 ノードの処理内容の記述方法

ノード内の処理を記述するには，ECMAScript を用いてコーディングを行う．ECMAScript によって記されたコードはコンパイラによって S91 バイトコードにコンパイルされる．ECMAScript には独自の拡張したシンタックスと組込みクラスとして，ファイルに仮想化する対象とするメソッドや変数と，それ以外のメソッドの扱いを区別する為の識別子である global があり，C -- ’ と同様でファイルに仮想化される．図 5 にソースコードの例を示す。

サーバ上の ECMAScript プログラムにおける，クライアントや他ノードとの通信用のファイルを利用するための組込みクラスを表 4 に示す．

ノードの周辺機器は S91VM によってファイルに仮想化されており，ファイル入出力を用いて周辺機器を利用する．これは，mount した先の RPC 対象の手続きや，その他のファイルに仮想化されているものにつ

GetTemp.ecma

```
public var a;        /*ファイルに仮想化されるグローバル変数*/
/*ファイルに仮想化されない*/
function GetAd() {
      var fp = new FileInputStrem(“node/dev/ad”);
      this.ad = new BufferReader(
                        new InputStreamReader(fp));
}

/*ファイルに仮想化される手続きを含む*/
function GetTemp() {
      /*ファイルとして仮想化される変数*/
      public var temp;

      /*ファイルに仮想化される変数に対応する手続き*/
      this.temp = this.ad * (変換式);
}

/*同一ノード内に記述されている手続きを継承*/
GetTemp.prototype = new GetAd();

/*ノードのマウントにより自ノード内に継承された手続きを継承*/
function GetWet() {
      public var wet = this.ad * (変換式);
}

/*GetAd2はマウントしているノード内のクラス*/
GetWet.prototype = new GetAd2();
```

**図 5** ノードのソースコードの例とノードの利用方法

**表 4** ECMAScript 用 File クラスのメソッドの設計

| 名前 | 機能 |
|---|---|
| Mount | 他のノードを mount する |
| Umount | 他のノードを umount する |
| FileInputStream | 通信用ファイルを開く |
| InputStreamReader | 通信先から値を取得する |
| BufferReader | バッファに読み込む |
| OutputStreamWriter | 通信先に値を送る |

```
Linux(Plan9)
modprobe 9p

～ノードのマウント～
mount -t 9p -o proto=tcp 192.168.5.141 /n/my_node/

～ノードの初期化～
echo new > /n/node/root/ctrl
～クラスディレクトリの生成～
cat GetTemp.s91 > /n/node/root/ctrl

～インスタンス化～
mkdir /n/my_node/node/GetTemp/Ins/my_Temp
mkdir /n/my_node/node/GetTemp/Ins/your_Temp

～RPCの実行～
cat /n/node/root/GetTemp/Ins/my_Temp/temp

～サブクラスの生成～
mkdir /n/node/root/GetTemp/GetTemp_sub

～他ノードへの継承～
echo mount 192.168.5.142 >
/n/node/root/GetTemp/GetTemp_sub/ctrl

～インスタンスの消去、資源の解放～
rm -r /n/node/root/GetTemp/Ins/your_Temp

～全クラス/インスタンスの消去、資源の解放～
echo kill > /n/node/root/ctrl
```

**図 6** ノードへのアクセス例

いても同様であり，ファイル入出力が他の資源への統一したアクセス方法である．

図 6 にノードの mount，S91 バイトコードのロード，インスタンスの初期化，RPC の実行，インスタンスの消去と資源の解放の流れを Unix のコマンドを利用した方法で記す．端末からのコマンドのみで RPC を実行できることが特徴であり，試作で用いた設計方針と同様である．

### 8.4 ノードの名前空間と継承

ノードが持つ名前空間はノード内で完結しており，ノード内に記述されているクラスは相互に利用可能である．ノード内のクラスはディレクトリに該当し，変数はファイルに該当する．変数に public 識別子がつき，ファイルに仮想化されて外部に公開されることが宣言されることで VM がファイルツリーに仮想化するクラスやメソッドを特定する．public 識別子を持つ変数を持つクラスはディレクトリに仮想化され，そのディレクトリ内のファイルとして変数が見える．これによって，ノードの手続きをファイルに仮想化し，ノードの利用者にファイルアクセスでのインタフェースを提供する．

ノードの継承は，mount 元にあたるノードの mount 先のディレクトリは何らかのクラスに該当するディレクトリであり，そこに他のノードを mount することで，mount したディレクトリの保持するクラスやメソッドを mount したノードからも参照可能にする．mount されたノードにはディレクトリが自動的に生成され，mount 元のディレクトリ内の内容が反映される．これにより，mount 先のノード内で継承元のクラスやメソッドが利用でき，ノード間を跨いでの資源の探索が可能になる．

ファイルの入出力は自分のノードのルートより下のみで実現可能で，ノードの上には辿れない．ファイルパスの絶対パスも自ノードのルートを起点に表現する．相対パスでは，ルートより上にさかのぼる場合はエラーとなる．これは，ノードを mount しているクライアントが 9P サーバを保持している確証がないからである．クライアント側の資源を活用したい場合，ノードの継承関係を利用し，スーパークラスの手続きを参照することで解決する．

### 8.5 VM の提供するファイル木

図 5 に示したノードのプログラムをもとに，コンパイラと VM が自動的に生成したファイルツリーが図 7 であり，四角はディレクトリ，丸はファイルを表している．また，各階層を繋ぐ線が普通の線のものは継承関係を，破線のものは通常のファイル階層を表している．

ノードにプログラムが載っておらず，何からも継承されていない状態では node ディレクトリの下には，周辺装置をファイルに仮想化したものと，他のノードの資源をファイル入出力で利用するための connection ディレクトリが格納されている dev ディレクトリが存在する．これは，ノード内のプログラムから共通して利用されるためノードのルートディレクトリにあたる node ディレクトリの直下のみに存在する．node ディレクトリ自体は空のクラスと解釈しており，コンソールファイルの役割の ctrl ファイルも最初から存在する．

それ以外のクラスディレクトリやグローバル変数は，ctrl ファイルを通して書き込まれたプログラムを元に S91VM が自動的に生成する．クラスディレクトリが生成されると同時に，クラスディレクトリ内にはそのクラスのインスタンスを格納する Ins ディレクトリと ctrl ファイルも生成される．Ins ディレクトリ内にディレクトリを作成する事がインスタンス化に該当し，インスタンスの名前は製作者に一任される．プログラム中のクラスの名前を元にクラスディレクトリが生成され，public 拡張子が付いていた変数の名前を元に手続きや変数が ins ディレクトリ内でファイルに仮想化される．

つまり，継承関係にあるディレクトリはクラスを表

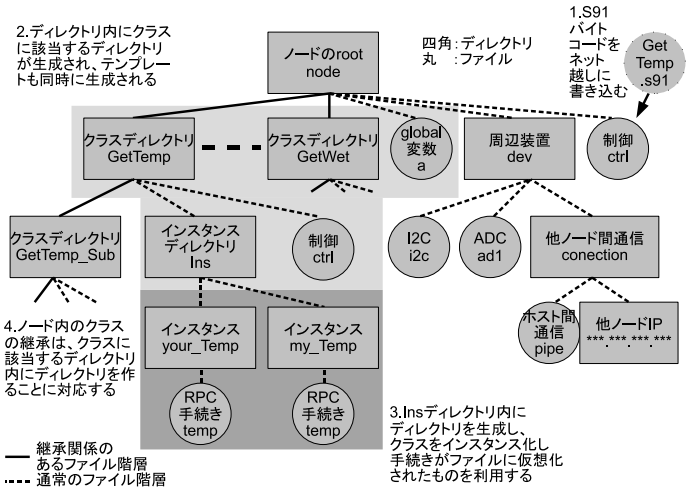


**図 7**　ECMAScript を利用した VM 提供ファイルツリー例

しており，それ以外のディレクトリは通常のファイルツリー中のディレクトリ同様，階層を意味する．ファイルはコンソールとしての機能を持つ ctrl ファイル以外は，何らかの資源を表している．ファイルはプログラム内容や，ハードウェアの資源に応じて S91VM が自動的に生成するので，利用者が作成する事はない．

## 9. おわりに

本研究では分散システムの通信プロトコルに 9P プロトコルを用い，ノードの手続きをローカル空間でのファイル入出力のように扱う，位置透過性の高い分散システムの構築法を提案した．ノード上の手続きをロードすることも，RPC の実行もファイルに対する入出力を通して行える．VM 上で動作するインスタンスを管理する為のインターフェースにもファイルを用い，Plan9 のような文字列により，VM の実行を制御できる．ソケットのような低水準な入出力を用いることなく，クライアントからもノードからもファイル入出力のインターフェースを用いて通信できる．また，提案した分散システムを実現するために，S91VM と C -- ' 言語の処理系の試作を行い，ノードで動作する処理系として，十分な実行速度を実現した．加えて，ECMAScript を利用して，ノード間の継承を有効にする提案手法により，個々のノードへのプログラミングを少ない手間で行うことができる．

主な課題は，VM の RAM 使用量を減らすことと、9P プロトコルの実装を VM 上で進め，すべての 9P プロトコルに応じた処理を実装することである．サーバとしての機能を拡充したのちには，クライアントとしての機能を持たせることで，9P プロトコルをサポートする処理系との連携を強化することである．ECMAScript を利用するコンパイラ，VM 同士の連携を含めた機能拡張も今後の課題である．

## 参 考 文 献


1) Bell-labs: Plan 9 from Bell Labs Fourth Edition , plan9.bell-labs.com/plan9/
2) Bell-labs: Plan 9 File Protocol, 9P , plan9.bell-labs.com/sys/man/5/INDEX.html, 2008.
3) Common Object Request Broker Architecture: Core Specification: Object Management Group, http://www.omg.org/cgi-bin/doc?formal/04-03-01.pdf
4) 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.
5) Crossbow: Wireless Module Portfolio, www.xbow.com/Products/productdetails.aspx?sid=156 .
6) vita nuova: Inferno, www.vitanuova.com/inferno/
7) SilentSystem: OS1 プログラミングマニュアル, www.silentsystem.jp/download/OS-1_Programing_06.pdf (2007).
8) Philip Levis, David Culler: Mate: a tiny virtual machine for sensor networks ACM SIGOPS Operating Systems Review Volume 36 pp.85-95, 2002.