# 装置組み込み用高速 Q3 エージェントプラットフォームの実現

登内敏夫† 中島震†

伝送装置，交換装置のマルチベンダ化の進展にともない，ITU-T のネットワーク管理標準である TMN の重要性が増してきた．しかし，TMN で規定される Q3 インタフェースの仕様は複雑で，その実装は，SNMP などの軽いプロトコルに比べて，速度性能が劣るといわれている．我々は高速な CMIS サービスとイベント通知が可能な Q3 エージェントを構築するためのエージェントプラットフォームを開発した．本エージェントプラットフォームは装置組み込みエージェント向けであり，ワンボード CPU 上で実行可能な小型高速エージェントを実現することができる．管理モデルの性質に応じた適切な実装を選択可能なオブジェクト指向フレームワーク方式を採用することにより，M-GET 秒速 400 オペレーション，M-EVENTREPORT 秒速 150 通知を達成した．

# Implementation of a Fast Q3 Agent Platform for Agents Embedded in Network Elements

Toshio Tonouchi† and Shin Nakajima†

Telecommunication Management Network (TMN) standardized by ITU-T becomes important because network elements developed by many vendors coexist in networks. A Q3 interface, which is a TMN interface between a manager and an agent, is complicated and burdensome, and a Q3 agent, therefore, is less efficient than an SNMP agent. We implemented an agent platform for a Q3 agent of high performance. The platform is used for developing an agent embedded in network elements, and the developed agent can run with high performance on one-board computer with a small amount of memories. We achieved 400 M-GET operations per second and 150 event notifications per second in sake of devised MIB data structure.

## 1. はじめに

Fiber-To-The-Home 時代の到来により，加入者網が急速に増加する．基幹網から加入者網までを OSI の枠組みで CMIP[1]を用いて統一的に管理するために，加入者網を構成するネットワーク機器（NE）向けの Q3 エージェントを開発する必要がある．本研究では，NE 組み込み型の Q3 エージェントプラットフォーム（PF）を開発することとした．Q3 エージェント PF を単一 CPU ボード上に実現し，NE と同一の筐体に組み込んだ形態を目指す．

基幹網に比べ，加入者網では非常に多くの NE が必要になることから，Q3 エージェントを含めた NE のコストが網全体のコストに影響する．Q3 エージェント用 CPU ボードに処理能力の高い高価な CPU を搭載することは，コストの面で困難である．

また，コストだけではなく，信頼性も Q3 エージェントプラットフォームに対する重要な要件である．そこで，不揮発記憶媒体としてハードディスクではなく，機械的な構成要素のないフラッシュRAM を用いることを検討している．フラッシュRAM の記憶容量あたりの価格はハードディスクに比べて数十倍になるため，大量のメモリを CPU ボードに搭載することが難しい．

さらに，Q3 エージェントが用いる管理プロトコル CMIP は，コンピュータネットワークで普及している管理プロトコル SNMP[2] に比べて，複雑な処理を必要とする．そこで，高速かつ省メモリ型の Q3 エージェント PF 実現方式の研究開発が必要となる．

## 2. TMN と Q3 エージェント

ISDN やインテリジェントネットワークなどが普及した現在，NE だけでなくネットワーク管理も高度化し複雑になっている．この管理基盤を定めているのが，国際標準化組織 ITU-T（International Telecommunication Union – Telecommunication Standardiza-

† NEC C&C メディア研究所
C&C Media Research Labratories, NEC Corporation

tion Sector）で標準化されている Telecommunication Management Network（TMN）である．TMN では管理対象・管理システムに関するインタフェースを定めることにより，相互接続性を実現している．このインタフェースに従えば，複数のベンダが開発した管理システムを相互接続できる．Q3 インタフェースはエージェント–マネージャ間を規定するインタフェースであり，本研究のエージェント PF は Q3 インタフェースを提供することにより，マネージャとのマルチベンダ相互接続性を実現する．

管理対象となる NE は管理対象オブジェクト（MO）としてモデル化される．MO インスタンス（MOI）は，包含木と呼ぶ木構造に配置され，MIB と呼ぶデータベースに格納される．MO は名前属性と呼ぶ属性を有する．名前属性の型情報（名前属性名）と値のペアの列である識別名（Distinguished Name: DN）を用いて，MOI の包含木上の位置を示す．

管理情報にアクセスするために CMIS[3] と呼ぶ 7 種類のサービスが提供されている．たとえば，M-GET サービスは MO の属性値を取得する．M-EVENTREPORT サービスはエージェントがマネージャにイベントを非同期的に通知する．CMIS のいくつかのサービスでは，スコープ・フィルタ処理により，複数の MOI を処理対象とすることができる．

このような CMIS を実現するアプリケーション層プロトコルとして CMIP，その通信データ形式として ASN.1[4),5)] が使用される[6]．

GDMO テンプレート[7] と呼ぶ仕様記述を用いて管理対象モデルを定義する．GDMO テンプレートにより，MO の構造（MANAGED OBJECT CLASS テンプレート）や MO 間の包含木上での関係（NAME BINDING テンプレート）などを記述する．

## 3. Q3 エージェントにおけるボトルネック

以下の要因が Q3 エージェントでの性能向上のボトルネックとなっている．本研究では，下記の 3 つのボトルネックを解決するためのアーキテクチャを提案する．

(1) 包含木の構造が複雑であるため，高速な検索処理を実現できない．包含木の検索キーである名前属性は様々なデータ型をとりうる．そのため，異なる型の属性値が混在し，効率の良い検索を行うことが困難である．

包含木中で一致する名前属性を探すには，まず，名前属性名どうしを比較し，次いで，名前属性値の比較を行う．このように 2 度の比較操作が必要であるため効率が悪い．

(2) MO は MO クラスごとに異なる振舞いを行う．たとえば，MO log は，イベントを logRecord として自らの直下の包含木に記録する．log は logRecord を一定数以上生成すると，古い logRecord を消去する（wrap 動作[8)]）．このように，log の直下の包含木は他の包含木とは異なる振舞いを提供する必要がある．

様々な振舞いをすべて共通のアーキテクチャコンポーネントで提供すると，処理効率が低下する．

(3) 管理モデルの抽象度が高いため，NE の状態と MIB の状態間のデータ転送・データ変換処理の負荷が高い．たとえば，NE の状態を MIB 内のデータ形式に変換する必要がある．我々はこれをビットマップ変換と呼んでいる[9]．MIB に転送した NE の状態をすべてビットマップ変換するとエージェントの性能が劣化する．

## 4. エージェントアーキテクチャ

本エージェント PF を中心とするエージェントの全体アーキテクチャを**図 1** に示す．

**エージェントアプリケーション（エージェント AP）** ネットワーク管理エージェントを実現するためのプログラム．装置・モデルごとに実装される．

**MIB PF**[10] MIB 構築用モジュール．MIB PF 上にエージェント AP を構築する．

**NEAE 1, NEAE 2**[9] NE と MIB をつなぐためのデバイスドライバ．NEAE 1 は MIB 側，NEAE 2 は NE 側に位置するドライバである．両者は UDP で交信し，NE–MIB 間の状態の一貫性を保持する．

**ISODE**[11] RFC 1006[12] を提供するプロトコルスタック．下位層は OSI ではなく，TCP/IP である．また，本研究では，ASN.1 コンパイラとして，ISODE が提供する ASN.1 コンパイラ Pepsy を使用している．

**Lynx OS**[13] POSIX 準拠のリアルタイム OS.

**NE** ネットワーク機器．NE は特に限定はしていない．X.721[14]，M.3100[15]，G.774[16] などの管理モデルを想定している．

**マネージャ** Q3 インタフェースを有するマネージャ．

図 1 中，破線矩形で囲まれた部分がエージェントであり，エージェントからエージェント AP を除いた，二点鎖線で囲まれた矩形がエージェント PF である．我々は，エージェント PF として，MIB PF，NEAE 1，NEAE 2 を開発した．また，エージェント PF の一部として，GDMO トランスレータ[17] などのエージェン

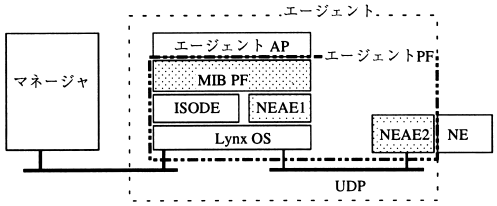


**図 1** 全体アーキテクチャ
Fig. 1 Modules in the architecture.

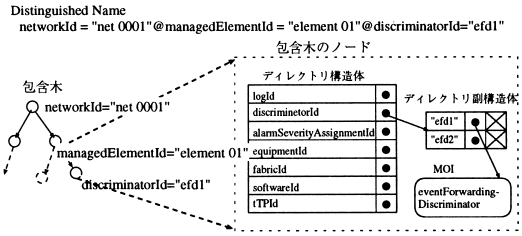


**図 2** ディレクトリ構造体とディレクトリ副構造体
Fig. 2 A directory structure and directory substructures.

ト AP 開発支援ツールを提供する．

本論文では，MIB PF のアーキテクチャを中心に述べる．MIB 中のデータ構造が，小型かつ高速なエージェントシステムを実現するためのキーポイントであるからである．本論文では以下の 3 点のアーキテクチャコンポーネントについて詳細に述べる．

(1) 包含木のデータ構造を工夫することにより，高速な包含木検索を実現する．
(2) 包含木のノードを，MOI の生成方法により分類し，最適な包含木ノードの実装を選択できるようにする．
(3) 属性値の変化の仕方に着目して，属性を分類し，最適な格納形式を選択できるようにする．格納形式の 1 つとして，リソース属性を導入し，NE–MIB 間のデータ転送の高速化とビットマップ変換処理を軽減する．

上記項目 (2)，(3) のように管理モデルに応じて適切なアーキテクチャコンポーネントを提供することにより，MO による性質の違いに対応した最適な実装を実現できる．最適な実装は，GDMO テンプレートを参照することで AP 設計時に選択できる．

本エージェント PF では，複数のアーキテクチャコンポーネントの導入によって生じるシステムの複雑化を克服するため，オブジェクト指向フレームワーク技術を採用した．これにより，アーキテクチャコンポーネントの違いを意識せずにエージェント AP を構築できる．

### 4.1 包含木：ディレクトリ構造体

包含木のノードを実現するために，名前属性名を格納する配列（ディレクトリ構造体）と，名前属性値を格納する配列（ディレクトリ副構造体）の 2 種類に分割したデータ構造を採用した．**図 2** は包含木の 1 ノードを表している．ノードは 1 つのディレクトリ構造体と複数のディレクトリ副構造体から構成される．

ディレクトリ構造体は，そのノードが保持する可能性のある MOI の名前属性名をキーとし，ディレクトリ副構造体へのポインタを値とする表である．各ノードで扱う可能性のある名前属性は NAME BINDING テンプレート[7]を参照することで AP 設計時に定めることができる．

ディレクトリ副構造体は，名前属性値と MOI へのポインタと下位のディレクトリ構造体へのポインタの 3 つ組のエントリからなる表である．

このノードから適切なエントリを選び出すには，以下の処理手順を行う．

(1) ディレクトリ構造体から，名前属性名が一致するエントリを選び出し，
(2) そのエントリが指すディレクトリ副構造体から，名前属性値が一致するエントリを選び出す．

ステップ (1) により，名前属性名が一致する MOI を保持するディレクトリ副構造体を選び出す．そのため，ステップ (2) では名前属性名の比較は不要で，名前属性値の比較のみを行えばよい．これにより，ベースオブジェクトを高速に検索できる．

### 4.2 3 種類のディレクトリ副構造体

3 章で述べた wrap 動作のように，包含木のノードはそれぞれ有する MO により異なる性質を持つ．MO によって性質が異なる包含木ノードを効率的に実現するため，3 種類のディレクトリ副構造体を用意した．

**静的ディレクトリ副構造体** 包含木ノード上で MOI を生成・削除しない場合に，もしくは，MOI の最大個数が決まっている場合に使用する副構造体．ディレクトリ副構造体を固定長配列で実現するため，ハッシュ関数を用いた高速な検索ができる．NAME BINDING テンプレートの CREATE，DELETE キーワードの有無により，MOI を生成・削除する可能性があるか否かを AP 設計時に決定できる．以下，本副構造体を静的副構造体と呼ぶ．

**動的ディレクトリ副構造体** 包含木ノード上で MOI を生成・削除する可能性がある場合に使用する副構造体．線形リストで実現する．線形リストに MOI を追加したり，削除したりすることで MOI 生成・削除を行うことができる．ただし，線形検索を行うため，検索速度は静的副構造体に比べ，低下する．以下，本副構造体を動的副構造体と呼ぶ．

**リング型ディレクトリ副構造体** wrap 動作を実装するための副構造体．固定長リングバッファ型のデータ構造を有しており，wrap 動作を効率的に実現することができる．たとえば長さ $m$ のリングバッファのエントリ 0 から $(m-1)$ に MOI を有する場合，次に生成された MOI は，最初に戻りエントリ 0 に格納する．その際，エントリ 0 に格納していた MOI を削除する．以下，本副構造体をリング型副構造体と呼ぶ．

### 4.3 3 種類の属性

属性値の変化の仕方により，属性を以下の 3 種類に分類し，それぞれの特徴に合った実装を提供する．

**通常属性** 以下に述べる共有属性やリソース属性以外の属性．通常属性は，MOI を実現する C++オブジェクトのメンバとして実装する．

**共有属性** MO クラスが同じならば，異なる MOI であってもつねに同じ属性値を有する属性．例として，属性 objectClass[14]があげられる．共有属性は，MO クラスを表す C++クラスの static メンバとして実装する．つまり，この MO クラスから生成した MOI はこの static メンバを共有する．これによりメモリ使用量を削減することができる．属性共有は文献 18) でも実現されている．

**リソース属性** 属性値が NE の状態を反映して変化する属性．たとえば，MO managedElement[15] の属性 operationalState は NE が動作可能（“enabled”）か否か（“disabled”）を示している．リソース属性の値は MIB 内の特定の連続メモリ領域（ビットマップ領域）に，NE 固有のデータ形式で配置されている．NEAE は NE の状態をそのままビットマップ領域に転送する．これにより，NE–MIB 間の高速なデータ転送が可能である[9]．また，リソース属性にアクセスがあった時点で，ビットマップ領域上のデータをビットマップ変換する．転送された NE の状態をすべてビットマップ変換せずに済むので，処理効率が向上する．

## 5. 性 能 評 価

以下の観点から性能評価を行った．

(1) 本アーキテクチャの速度性能の測定．
各 CMIS サービスごとの速度性能を計測した．

(2) アーキテクチャコンポーネントによる速度性能とプロセスサイズの計測．
各アーキテクチャコンポーネントにより，速度性能・プロセスサイズがどの程度影響を受けるかを計測した．

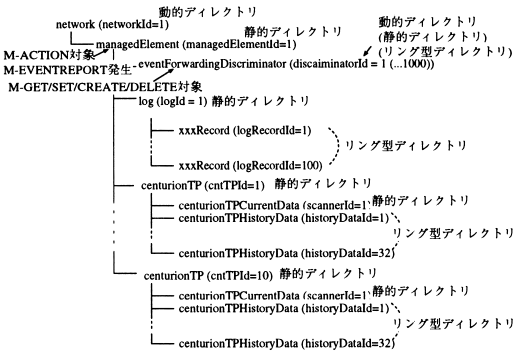


**図 3** 評価対象モデルの包含木
Fig. 3 A containment tree for the evaluation.

### 5.1 評 価 方 法

以下の方法で速度性能を測定した．

**評価システム** 本 PF で構築したエージェントと，簡易マネージャを異なるワークステーション上に実装した．両者およびプロトコルアナライザをハブを介して，イーサネット（10 BasseT）で接続した．本評価システムでは NE を接続する代わりにソフトウェアで疑似 NEAE を構築し，NE をエミュレーションした．

エージェントを実装したワークステーションの性能は以下のとおり．

**CPU** Ultra Sparc I（143 MHz）
**オンチップキャッシュ** 16 KB
**セカンドキャッシュ** 512 KB
**SPECint92** 215
**メモリ** 96 MB

**評価モデル** M.3100[15]に基づき，エージェントを構築した．**図 3** に評価対象モデルの包含木を示す．各 MOI の右横にどのディレクトリ副構造体を使用したかを示す．

**速度性能計測** プロトコルアナライザで CMIS 要求から CMIS 確認までの時間（応答時間）を計測した．

**プロセスサイズ** UNIX の ps コマンドでプロセスの仮想記憶サイズを計測し，必要記憶容量と見なした．

### 5.2 評 価 結 果

#### 5.2.1 基 本 性 能

**表 1** は本 PF および OSIMIS[19]の CMIS 操作に要する応答時間を計測した結果である．スコープとして baseObject，フィルタなしで応答時間を計測した．

M-ACTION, M-EVENTREPORT を除く各 CMIS 操作は図 3 で示すように eventForwardingDiscriminator（eFD）に対して行った．M-ACTION は man-

**表 1** 各 CMIS 操作での応答時間（msec）
Table 1 Response times (msec) of CMIS operations.

| | GET | SET | ACTION | CREATE | DELETE |
|---|---|---|---|---|---|
| 本 PF | 1.69 | 1.66 | 1.70 | 1.83 | 1.93 |
| OSIMIS | 1.85 | 1.81 | — | 1.99 | 2.04 |

**表 2** M-EVENTREPORT イベント通知時間（msec）
Table 2 Response times (msec) of M-EVENTREPORT.

| logRecord 蓄積 | イベント通知時間 | イベント/sec |
|---|---|---|
| あり | 6.3 | 158.5 |
| なし | 5.6 | 177.4 |

**表 3** eFD 数とプロセスサイズ（KB）の関係
Table 3 Process sizes (KB).

| #eFD | 0 | 100 | 500 | 平均 |
|---|---|---|---|---|
| 本 PF | 3,648 | 3,816 | 4,552 | 1.81 |
| OSIMIS | 3,112 | 3,272 | 3,668 | 1.14 |

agedElement を管理対象オブジェクトとし，ログファイルへの書き込みを行う☆．本アクションでは ACTION-INFO，ACTION-REPLY とも定義されていない．

ただし，本 PF と OSIMIS では評価モデルが異なるため，本 PF では包含木 3 段目の MO に対し GET/SET/CREATE/DELETE を行っているのに対し，OSIMIS では 2 段目の MO に対し行っている．

表 1 を見ると，すべての CMIS サービスで本 PF の方が OSIMIS より速度性能が良いことが分かる．

**表 2** は M-EVENTREPORT の送信時間を計測した結果である．log が logRecord を生成蓄積する場合としない場合の 2 種類に分けて，100 回の M-EVENTREPORT 送信の平均時間を計測した．本処理時間は，疑似 NEAE が疑似的に 100 回発生した障害発生を受けて，エージェントが managedElement からのイベントとして M-EVENTREPORT を発生するのに要する時間である．本評価では，eFD は 1 つ存在し，イベントをフィルタせずマネージャに通知する．

**表 3** は eFD の数とプロセスサイズを本 PF と OSIMIS を比較した結果である．本 PF では eFD 1 個を 1.81 KB で実現しているのに対し，OSIMIS は 1.14 KB で実現している．このようにメモリ効率は OSIMIS の方が良い．

**表 4** はスコープの種類ごとに M-GET 操作を行い，スコープの処理速度に与える影響を計測した結果である．「#MOI」はスコープの検索対象となった MOI 数を示す．「当初応答」は CMIS 要求から最初の連結応答確認までの時間を示す．「連結応答」は連結応答確認から次の連結応答確認の平均時間を示す．「平均」は総応答時間を連結応答の個数で割った値である．

☆ 本評価システムではハードディスクを搭載している．

**表 4** スコープの種類による M-GET 連結応答時間（msec）
Table 4 Response times (msec) of M-GET with scopes.

| スコープ | #MOI | 当初応答 | 連結応答 | 平均 |
|---|---|---|---|---|
| baseOnly | 1 | — | — | 2.4 |
| firstLevel | 1 | 9.0 | 39.4 | 46.4 |
| individualLevel | 12 | 7.5 | 5.4 | 6.0 |
| baseToNth | 14 | 5.0 | 2.5 | 2.9 |
| wholeSubtree | 444 | 9.0 | 1.8 | 5.1 |

**表 5** フィルタ長による M-GET 連結応答時間（msec）
Table 5 Response times (msec) of M-GET with filters.

| | 全 MOI 数 | 該当 MOI 数 | 平均 |
|---|---|---|---|
| フィルタなし | 124 | 124 | 4.6 |
| フィルタ長=1 | 124 | 100 | 5.9 |
| フィルタ長=2 | 124 | 100 | 6.3 |
| フィルタ長=3 | 124 | 100 | 6.4 |

**表 6** 属性の実装による応答時間（msec）
Table 6 Response times (msec) for attribute implementations.

| 操作 | 通常属性 | リソース属性 | 共通属性 |
|---|---|---|---|
| M-Get | 2.4 | 2.6 | 2.1 |

TCP のウィンドウサイズ枯渇によるフロー制御パケットが発行されているため，連結応答時間はかなりバラツキがある．フロー制御パケットが発行されると，余分な通信が発生し性能が劣化する．多くの送受信が発生する連結応答において，フロー制御パケットが多数発行されている．

**表 5** は，フィルタが速度性能に与える影響をフィルタ長に応じて計測した結果である．この計測では network をベースオブジェクトとし，スコープ wholeSubtree を使用した検索をする．「全 MOI 数」は MIB 内の MOI の総個数を示す．「該当 MOI 数」はフィルタに合致した MOI の個数を示す．「平均」は MOI 1 個あたりの平均連結応答時間を示す．

### 5.2.2 属性の実装によるアクセス速度と記憶容量

**表 6** は，属性の実装方法ごとの M-GET 応答速度を示す．各属性はそれぞれ同程度の処理速度を示している．リソース属性と通常属性で応答速度の差がほとんどないことより，ビットマップ変換による負荷を抑えることができたことが分かる．

共通属性がもたらす使用メモリ削減効果を，eFD の objectClass 属性を通常属性で実装した場合と共通属性で実装した場合で比較した．**表 7** は eFD の数を増加させて，プロセスサイズを計測し，両者を比較した結果である．「#MOI」は MIB 内の全 MOI 数を示す．

**表 7** 共通属性によるプロセスサイズ削減効果（KB）
Table 7 Process sizes (KB) with/without shared attributes.

| #MOI | 24 | 5,024 | 10,024 | 30,024 | 平均 |
|---|---|---|---|---|---|
| 通常属性 | 3,648 | 13,272 | 22,896 | 61,296 | 1.92 |
| 共通属性 | 3,648 | 12,600 | 21,536 | 57,296 | 1.79 |
| 削減効果（%） | 0.00 | 5.06 | 5.94 | 6.53 | |

「平均」は MOI 1 個あたりの記憶容量を示す.「削減効果」は通常属性で実装した記憶容量のうち，共通属性で実装するとどれぐらいの割合（%）で記憶容量を削減できたかを示す.

表 7 から分かるとおり，MOI 数が増えると，objectClass を共有する MOI も増加するため，共通属性によるメモリ削減効果が向上する.

### 5.2.3 ディレクトリ副構造によるアクセス速度

ディレクトリ副構造体の実装方法の違いにより，どの程度処理速度が異なるかを eFD を含むディレクトリ副構造体を計測対象とし，M-CREATE，M-GET，M-DELETE で計測した.

M-CREATE では，100 回 M-CREATE を発行し，MOI を 100 個生成し，平均応答時間を計測した．静的副構造体，リング型副構造体では，あらかじめディレクトリ副構造体に MOI 100 個分の空き領域を確保し，MOI を生成した.

M-GET では，スコープを baseObject とし，フィルタなしで応答時間の計測を行った．ただし，動的副構造体では，検索対象の MOI の位置により速度性能が変化する．そこで，副構造体内の線形リストの真中にある MOI を M-GET した.

M-DELETE では，M-CREATE で作成した MOI 100 個を削除し，平均応答時間を計測した.

リング型副構造体では，wrap 動作を実現するため，つねに MOI をリングバッファ上の連続領域に配置している．MOI が削除されると，空き領域を埋めるように MOI を移動させる（コンパクション）．コンパクションの有無が処理速度に影響する．そこで，MOI 削除の順番はランダムに行った.

**表 8** は評価結果である．M-CREATE，M-DELETE とも，静的副構造体は動的副構造体より処理が速い.

一方，M-DELETE では，リング型副構造体は他の副構造体と同等の速度性能が示された．コンパクションによる速度性能の劣化は本評価では現れなかった.

### 5.2.4 実 NE（OC-3）を使用した評価

本項では，実 NE を接続し評価を行った．実 NE として，ATM（OC-3）を接続し，単一 PowerPC ボードに Q3 エージェントを実装した.

**表 8** ディレクトリによる応答時間（msec）
Table 8 Response times (msec) for directory implementations.

| | M-CREATE | M-GET | M-DELETE |
|---|---|---|---|
| 静的副構造体 | 3.3 | 2.7 | 2.4 |
| 動的副構造体 | 3.5 | 2.8 | 2.4 |
| リング型副構造体 | 2.3 | 2.7 | 2.4 |

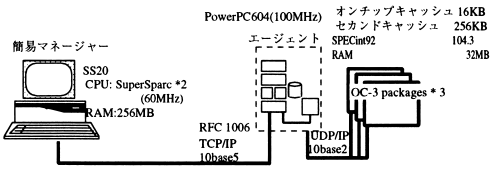


**図 4** PowerPC + OC-3
Fig. 4 Agent on PowerPC connected with OC-3.

**表 9** OC3 を接続した場合の速度性能（msec）
Table 9 Performance on PowerPC board.

| | M-GET | M-SET | M-EVENTREPORT |
|---|---|---|---|
| Sparc | 4.910 | — | 6.309 |
| PowerPC | 6.165 | 162.8 | 21.82 |

PowerPC システム構成およびスペックを**図 4** に示す.

**表 9** に評価結果を示す.「Sparc」は前節の評価で用いたワークステーションに実装したエージェントである．M-EVENTREPORT サービスでは Sparc 版に比べて 3 倍以上の応答時間を要している．これは，本 OC-3 装置ではこれ以上の速度でイベントを発生していないためである．言い換えれば，本 OC-3 装置ならば，エージェントが 21 msec より短い時間でイベントを送信する性能を有していれば速度性能は十分であるといえる．ちなみに，NE の代わりにプロトコルアナライザで疑似的にイベントを発生した場合，M-EVENTREPORT を約 150 イベント/秒で発行することができた.

PowerPC 版で M-SET に処理時間を要しているのは，NE 上にある属性に対して M-SET を行っているためである．このような属性をリソース属性の中でもオンデマンド属性と呼ぶ．エージェントはオンデマンド属性に対する M-SET 操作が発生すると，NEAE を通し NE に状態変化要求を通知する．そのため，ビットマップ領域に存在するリソース属性や通常属性よりもアクセス時間を要する.

## 6. 議　　論

OpenView[20] をはじめとする SNMP ベースの管理システム PF と同様に，TMN 関係の PF は，様々な製品が実現されている[21)~23)].

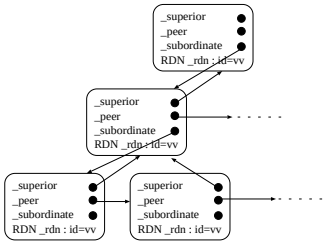


**図 5** OSIMIS の包含木データ構造
Fig. 5 Containment tree data structure of OSIMIS.

OSIMIS はソースコードが公開されている TMN 用エージェントツールキットである．OSIMIS の包含木構造と本 PF の構造を詳細に比較し，本 PF が高速な包含木検索を実現していることを詳細に述べる．

OSIMIS のディレクトリ構造は包含木上同じ高さの MOI をつなぐ線形リスト（`_peer` メンバ）と，その上位下位の MOI をつなぐポインタ（`_superior`, `_subordinate` メンバ）で構成されている（**図 5**）．このようなデータ構造では，包含木ノードから下位のノードを得るには，線形検索をする必要がある．たとえば，包含木ノードが有する $k$ 種類の名前属性が，それぞれ $n$ 個 MOI を有していた場合，平均 $kn/2$ 回検索する必要がある．これは，提案方式の静的ディレクトリなら 2 回，動的ディレクトリならば，$(1+n/2)$ 回に比べて検索時間が長い．

OSI ディレクトリの管理方式として，木構造のデータ型をルートから順次たどるのではなく，ハッシュ表を使用する方式も提案されている[24]．この方式では，木構造の中間にあるデータ構造に対しても $O(1)$ で検索できる．このようなハッシュ方式は，ハッシュ表が固定長であることを前提とするので，MOI の生成，削除を実装するのが難しい．

文献 24) で，リニアハッシュなど，ハッシュ表を稼働時に伸長させるアルゴリズムの適用に関する言及があるが，リニアハッシュはハッシュ表を 2 の冪乗で伸長するので，効率的でない．

また，文献 24) の方式では，スコープ処理実現のために，ハッシュ表情報のほかに，包含木の木構造のデータ構造も管理する必要がある．これは記憶容量の増加を招く．

次に，ディレクトリ副構造体をモデルに応じて分類したことのメリットを論じる．表 8 で示されるように，リング型副構造体は，MOI 生成処理を高速に実行できる．リング型副構造体は logRecord を格納することを想定している．NE に障害が発生した場合，一時に大量のイベントが発生し，logRecord も大量に生成される．このように logRecord は他の MO に比べて高速な生成処理が求められる．logRecord 格納にリング型副構造体を用いることで生成処理を高速化することができる．

次に，モデルに応じた属性の実装を提供することに関して議論する．表 7 で示すように，共通属性によるメモリ削減効果を確認した．この評価では，objectClass 属性のみを共通属性で実装した．他にも packages 属性や nameBinding 属性なども共通属性として実装でき，さらなる記憶容量の削減が期待できる．

本論文で述べたような属性を性質により分類し，最適な実装構造を行う研究はいくつかある．たとえば，文献 25) では，属性を値不変の属性（Static），マネージャから変更はしないが，値が変化しうる属性（Psudo Static，Dynamic），M-SET により値変更する属性（GET-SET）の 4 種類に分類し，その分類に応じたロッキングプロトコルを施すことにより，並列性の高いトランザクションを実現する．

一般のデータベースでは，その格納するデータの性質に触れずに一般的に処理することが多い．しかし，MIB では，管理モデルによりその格納するデータ（属性）の性質を AP 設計時に想定できるため，モデルの性質による効率的な格納方式をとりうる．

最後に NE–MIB 間のデータ転送・変換について議論する．表 6 で示されるように，オンデマンド属性を除くリソース属性は通常属性と同等の CMIS 応答性能を実現している．また，4.3 節で示したように NE–MIB 間の高速転送や不要なビットマップ変換の回避を実現している．

AP から見ると，オンデマンド属性もリソース属性と同様に実現されている．両者の実装の差異はビットマップ変換処理ルーチンが吸収する．つまり，オンデマンド属性の場合，変換処理ルーチンは NEAE を通し NE にアクセスするが，それ以外のリソース属性の場合，ビットマップ領域にアクセスする．

## 7. ま と め

我々は装置組み込みを目指した Q3 エージェント PF を開発した．本エージェント PF では，ディレクトリ構造を工夫することと，管理対象モデルの性質に応じたアーキテクチャコンポーネントを提供することにより，Q3 実装でのボトルネックを解消し，省メモリと高速性を実現を実現した．GDMO テンプレートに記述された管理モデルを参照することにより，AP 設計時に適切なアーキテクチャコンポーネントを選択することができる．

表 7 で示されるように本 PF での必要メモリは MOI

10,000 個で 22 MB を占める．MOI 30,000 個を格納すると 60 MB を超え，5.2.4 項で述べた 32 MB 搭載の PowerPC での実装は困難である．OSIMIS でも表 3 で示すされるように，32 MB では MOI を 30,000 個格納することは困難である．MIB 内のデータでは，ディレクトリデータ構造やクラス情報よりも属性などの ASN.1 データが記憶量の多くを占める．本 PF，OSIMIS ともに ASN.1 コンパイラ Pepsy を使用しているため，ASN.1 データに関しては同等の記憶容量を占める．そこで，ASN.1 コンパイラを置き換えることで，ASN.1 データが占める記憶容量を削減することが今後の課題である．

## 参 考 文 献


1) ITU-T: X.711: Common Management Information Protocol Specification for ITU-T Applications (1991).
2) IAB: RFC 1157: Simple Network Management Protocol (SNMP) (1990).
3) ITU-T: X.710: Common Management Information Service Definition for ITU-T Applications (1992).
4) ITU-T: X.208: Specification of Abstract Syntax Notation One (ASN.1) (1988).
5) ITU-T: X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) (1988).
6) 大鐘久生：TCP/IP と OSI ネットワーク管理，SRC ハンドブック (1993).
7) ITU-T: X.722: Guidelines for the Definition of Managed Object (1992).
8) ITU-T: X.735: Log Control Function (1991).
9) 竹居正浩，桐葉佳明，中井正一郎：装置組み込みエージェントにおける管理情報転送方式，電子情報通信学会全国大会 (1996).
10) Tonouchi, T., Fukushima, T., Manki, A. and Nakajima, S.: An Implementation of OSI Management Q3 Agent Platform for Subscriber Networks, *ICC '97*, Vol.2, pp.889–893 (1997).
11) Rose, T.: The ISO Development Environment: User's Manual.
12) IAB: RFC 1006: OSI transport services on top of TCP (1991).
13) Lynx Corp.: Welcome to Lynx Real-time Systems, http://www.lynx.com/.
14) ITU-T: X.721: Structure of Management Information Part 1: Management Information Model (1991).
15) ITU-T: M.3100: Generic Network Element Information Model (1995).
16) ITU-T: G. 774: Synchronous Digital Hierarchy (SDH) Management Information Model (1992).
17) 登内敏夫，中島 震：出力コードを容易に変更可能な GDMO トランスレータ，ソフトウェア科学会第 14 回大会，pp.1–4 (1997).
18) 佐々木忍ほか：リアルタイム OS を用いた TMN エージェントプラットフォーム，信学技法，IN94-120 (1994).
19) Pavlow, G.: The OSI Management Information Service User's Manual (ver.1) (1993).
20) Packard, H.: HP OpenView, http://www.openview.hp.com.
21) DSET Corp.: Agent ToolKit, http://www.dset.com/tmn/agent_toolkit1.html (1998).
22) SUN Micro Systems: Solstice TMN Toolkit, http://www.sun.co.jp/software/products/ENP/telecom/TMNtoolkit.html (1999).
23) Shimizu, T., Yoda, I. and Fujii, N.: Implementing and Deploying MIB in ATM Transport Network Operations System, *ISINM 95* (1995).
24) 西山 智，横田英俊，小花貞夫，鈴木健二：OSI ディレクトリ情報ベース（DIB）のためのハッシュを用いた高速名前解析処理方式，情報処理学会論文誌，Vol.35, No.12, pp.2762–2773 (1994).
25) Nakai, S.: MIB Desigin for Network Management Transaction Processing, *Integrated Network management III*, pp.97–108 (1993).


(平成 11 年 2 月 12 日受付)
(平成 12 年 2 月 4 日採録)

**登内 敏夫**（正会員）

平成 4 年東京大学大学院理学系研究科情報科学専攻修士課程修了．同年日本電気（株）入社．現在 NEC C&C メディア研究所勤務．ネットワーク管理を対象領域としたソフトウェア基盤やソフトウェア開発用言語処理系の研究開発に従事．

**中島 震**（正会員）

昭和 56 年東京大学大学院理学系研究科物理学専攻修士課程修了．同年日本電気（株）入社．現在 NEC C&C メディア研究所勤務．分散オブジェクト指向技術，フォーマルメソッドの適用に関する研究開発に従事．昭和 63 年〜平成元年米国オレゴン大学訪問研究員．平成 4 年より東京都立大学工学部非常勤講師を兼務．平成 11〜12 年度日本ソフトウェア科学会理事．