マルチメディア通信と分散処理 75－6
（1996. 2. 29）

網管理エキスパートシステムのための SNMP通信スタブ生成ツールの実装

松本 一則　橋本 和夫　谷戸 文廣　小花 貞夫

KDD研究所

〒356 埼玉県上福岡市大原 2-1-15

E-mail: {matsu, kh, yato, obana}@lab.kdd.co.jp

LAN機器等の管理に広く使われているSNMP (Simple Network Management Protocol) 上で高度な網管理を実現するため，エキスパートシステム (ES) からSNMPを利用可能とする通信スタブを管理対象 (MO) の定義から自動生成するツールを実装した．生成する通信スタブは，(1)MOの種類毎に対応したWME (Workibg Memory Element) の型宣言，(2) 取得したMOの情報をESのワーキングメモリに自動的に反映する管理操作関数，(3) 実際のMOとWMEとの一貫性を管理するインスタンス管理部からなる．本ツールが生成するプログラムの規模はMOの種類 (クラス) あたり1.3Kステップであること，また，生成した通信スタブは実用的な処理時間で属性の設定と取得を行えることを確認し，本ツールの有効性を検証した．

An Implementation of SNMP Communication Stub Generator for Network Management Expert Systems

Kazunori MATSUMOTO　Kazuo HASHIMOTO　Fumihiro YATO　Sadao OBANA

KDD Research and Development Laboratories

2-1-15, Ohara, Kamifukuoka-shi, Saitama 356 JAPAN

E-mail: {matsu, kh, yato, obana}@lab.kdd.co.jp

We have developed a stub generator which enables expert systems to use SNMP (Simple Network Management Protocol), widely used protocol for LAN management. The generated stub consists of (1) type declaration of WME (Working Memory Element) for each Managed Object (MO), (2) management operation functions which reflect collected data of MO into working memory, (3) instance management module which keeps consistency between actual MOs and WME. We ensured effectiveness of the stub generator from the point of program size and performance of generated stub.

1 はじめに

網管理の分野では，TMN(Telecommunication Management Network) や SNMP(Simple Network Management Protocol) などの標準プロトコルを用いた異種システムの統合管理が進められている．今後は，障害状況の推定や原因分析などの高度な網管理を実施するため，エキスパートシステム (ES) の導入が重要となってくる．例えば，障害管理では，複数の管理対象 (MO: Managed Object) の情報を組み合わせて，障害の診断を行う必要がある場合もある．この場合には，ある条件を満たすMOの集合を高速なパターンマッチング機能を用いて列挙し，また，列挙条件を柔軟に設定することができるESが適している．

網管理システムを構築する場合，管理対象となる機器の種類ごとに定義されるMOに対して管理操作を可能とするプログラム (通信スタブ) が必要となるが，MOの種類が多いシステムでは通信スタブの作成が多大な作業となる．このため，MOの定義情報から，通信スタブを生成する手法が提唱されている．既に，C言語を対象とした通信スタブ生成ツール [1],[2] は報告されているが，ES向きの言語を対象とした通信スタブに関しては，実装等の報告がない．

筆者らは，SNMPを用いて高度な網管理を実現す

るため，ESからSNMPを利用可能とする通信スタブをMOの型定義から自動生成するツールを実装した．本稿では，本ツールの実装とその評価結果について報告する．

2 網管理ESのための通信スタブと自動生成の必要性

網管理システムでは，網を構成する機器の特定パラメータに対する値の取得や設定といった基本的な操作を実行するエージェントと，これらの機能を組み合わせて網管理を行うマネージャとで構成される．マネージャとエージェントとの情報交換には，SNMP等の網管理プロトコルが使用される．網管理ESはESが提供する知識処理によって，高度な網管理を行うマネージャ用アプリケーションである．以下では，網管理ESが網管理プロトコルを使ってエージェントと通信するために必要なプログラムを網管理ES用通信スタブ(以下では通信スタブと呼ぶ)として位置付け，その機能と自動生成の必要性を述べる．

2.1 網管理ES用通信スタブの機能

ESは，パターンマッチングの対象となるデータをワーキングメモリ(WM)と呼ばれる記憶領域にワーキングメモリ・エレメント(WME)として保持する．高速なパターンマッチングアルゴリズム[3]を実現するため，一般的にWMEは特別なデータ型を用いて実装され，その使用に関して以下の制約が設けられている．

- パターンマッチングの高速化のため，WMEのフィールドで使用可能な型にポインタが許されない．
- WMEの生成，修正，削除には特別に用意された関数で行う．

一方，網管理プロトコルを扱うためのライブラリ等が提供しているXMP/XOM[4]等のAPIでは，管理操作関数やデータ型がパターンマッチングを考慮したものでない．このため，通信スタブは，上記APIのデータ型とWMEのデータ型を相互変換し，ESから網管理プロトコルを用いるための操作関数を提供する．

2.2 通信スタブの自動生成の必要性

網管理ESでは，MOの情報をWMEに変換してパターンマッチングを行なうため，MOのクラス毎のWMEへの変換と管理操作を行うための関数を，通信スタブがアプリケーション開発者に提供する必要がある．このため，MOのクラスが多数存在したり，新たな通信機器が頻繁に導入される網の場合，膨大な通信スタブのプログラムを頻繁に追加・変更しなければならない．そこで，MOの定義からの通信スタブの自動生成が望まれる．

3 通信スタブ生成ツールの実装

3.1 基本方針

(1) 網管理プロトコルとしてLAN機器の管理に広く普及しているSNMPを用いる．

(2) ESの構築に広く用いられているOPS83をES用言語とする．

(3) 網管理プロトコルを使用するためのAPIにXMP/XOMを用いる．

(4) MO定義にはOSI管理のGDMO(Guidelines for the Definition of Managed Objects)[5]を用いる．

特に，(3), (4)については，将来的なOSI管理への拡張を考慮し，網管理プロコル用APIにSNMPとOSI管理プロトコルであるCMIS/CMIPとを包含した仕様であるXMP/XOMを，また，MO定義の記述にGDMOをそれぞれ選択した．

3.2 通信スタブ生成ツールの概要

通信スタブ生成ツールは，GDMOで記述されたMO定義から，(1)MOの型宣言，(2)管理操作関数，(3)インスタンス管理部を生成する．ただし，通信スタブの一部として本ツールが提供する(4)通信環境管理ライブラリについては，MOの定義内容に依存しない(図1参照)．図中の通信スタブ内の( )はプログラミング言語を示す．網管理ESのプログラムは，生成された通信スタブ，WMの管理とパターンマッチング処理を行うOPS83の実行用ライブラリ，XMP/XOMのライブラリ，および知識処理のユーザプログラムをリンクして構成する．

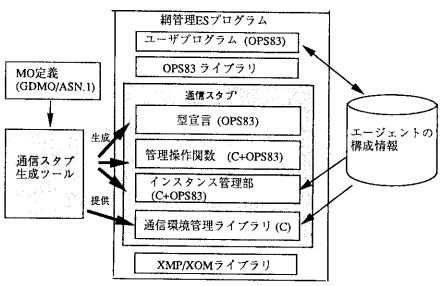

図 1: 網管理ESのための通信スタブ生成ツール

MO の型宣言: GDMO で記述した MO 定義に対応する，OPS83 における WME のデータ型定義である．OPS83 では，パターンマッチングの対象となる WME は，エレメントと呼ばれるデータ型で宣言する．

管理操作関数: MO に対し設定要求 (set-request)，取得要求 (get-request) や取得要求継続 (get-next-request) を行う関数と，設定応答 (set-response)，取得応答 (get-response) やトラップ (trap) を受信した際に WME の値を自動的に更新する関数である．

インスタンス管理部: 管理対象となる機器が新たに追加された時にその機器に対応する WME を生成する等の処理を行い，MO と WME との一貫性を保持する機能モジュールである．インスタンスを管理する上で必要となるエージェント毎の構成情報は，外部ファイルから取得する．

通信環境管理ライブラリ: エージェントとのセッション確立，シープ (トラップの受信条件) の設定等の SNMP 通信の環境設定をユーザプログラムから行うためのライブラリである．また，非同期通信処理を行うための，ユーザプログラムと通信スタブを交互に呼び出すデスパッチャを提供する．

4 通信スタブの生成

4.1 型宣言の生成

GDMO におけるデータ型定義に使われる ASN.1 と OPS83 の基本的な型の対応付けを表 1 に示す．ただし，SNMP では，ASN.1 の OPTIONAL，DEFAULT，SET 型，CHOICE 型は使用しないため，OPS83 との対応付けは行なっていない．

表 1: ASN.1 と OPS83 の型の対応付け

ASN.1 の型	OPS83 の型
BOOLLEAN	logical
ENUMERATED	symbol
INTEGER	integer
NULL	integer
REAL	real
CHARACTER STRING	
(32 文字以下)	symbol
(128 文字以下)	name
(129 文字以上)	array(char:文字数)
OCTET STRING	CHARACTER STRING と同様
SEQUENCE (SEQUENCE OF)	record (record の配列)

OPS83 では文字列を表わす name 型よりも文字列をハッシュ化した symbol 型の方が高速なパターンマッチングが可能である．しかし，全ての文字列を symbol 型で定義するとハッシュテーブルのサイズが膨大になってしまう．そこで通信スタブ生成ツールでは，ハッシュテーブルサイズを抑えるため，インスタンスの識別等に用いられてパターンマッチングの対象となる短い文字列のみを symbol 型として扱い，パターンマッチングの対象とならない長い文字列は，記憶容量を節約できる array に対応させた．

OPS83 では，C++のように上位クラスから属性を継承させることで記述量を減らすことができない．そこで，上位クラスから継承する属性は，ローカルな属性同様，該当クラスを定義したエレメントのフィールド中に宣言することとした．このため，通信スタブ生成ツールは，型宣言生成の際に，GDMO で定義されたクラス間の継承関係を解析して，継承すべき属性を全て列挙する．

また，後述する管理操作関数やインスタンス管理部の処理では，MO のインスタンスを一意に識別する識別子が必要となる．このため，MO に対応する OPS83 の型宣言には，識別名とインスタンス ID を表わす label と id と呼ぶフィールドをそれぞれ設けた．

(1) GDMO で記述した MO 定義

```
internetSystem MANAGED OBJECT CLASS
  DERIVED FROM "Rec.X.721|ISO/IEC ":top;
  CHARACTERIZED BY internetSystemPkg  PACKAGE
    ATTRIBUTES
      internetSystemId GET,
      sysDescr GET,  sysObjectID GET,
      sysUpTime GET,  sysContact GET-REPLACE,
      sysName GET-REPLACE,
      sysLocation GET-REPLACE,  sysServices GET;
    NOTIFICATIONS  internetAlarm;;;
REGISTERED AS { 1 3 6 1 2 1 1 };

sysContact  ATTRIBUTE
  --  本来は displayString
  WITH ATTRIBUTE SYNTAX OCTET STRING SIZE(32);
  -- 32 文字の文字列に変更
REGISTERED AS { 1 3 6 1 2 1 1 4 };
```

(2) 生成した OPS83 の型宣言

```
type internetSystem = element (
  label:symbol;  /* インスタンス識別子名 */
  id:integer;  /* インスタンス識別子番号 */
  /* top の属性 */
  objectClass:symbol;
  nameBinding:symbol;
  /* 属性 */
  internetSystemId:integer;
  sysDescr:name;
  sysObjectID:oidTYPE;
  sysUpTime:integer;
  sysContact:symbol;
  sysName:name;
  sysLocation:name;
  sysServices:integer; );
```

図 2：MO 定義と生成した型宣言の例

図 2 は，MO 定義と表 1 の対応付けに従って生成される型宣言の例である．同図では，MO 定義で 32 文字のオクッテト文字列とした sysContact が，生成される型宣言では symbol 型として宣言されている．また，上位クラス top から継承した属性 objectClass と

nameBinding が internetSystem の中で型宣言されている.

4.2 管理操作用関数の生成

図 3(1) は生成される OPS83 の管理操作関数であり，要求 ID の取得関数，要求 ID の解放関数，属性毎に生成される SNMP の操作に対応した，設定要求関数，取得要求関数，取得要求継続関数である．図 3(2) は管理操作関数で使用する要求 ID 等のデータである．

(1) 管理操作関数

```
/* 要求 ID の取得関数 */
function CI_get_id()  : integer;
  要求 ID として使用可能な番号を返す
/* 要求 ID の解放関数 */
function CI_free_id(&id: val integer): CIResult;
  指定された要求 ID を再利用可能にし，不要となるトランザクション管理フラグを WM から削除する．
/* 設定要求関数 */
function CI_set_クラス名_属性名 (
    &request_id: val integer,
    &instance_id: val integer,
    &attr_val: val 属性型): CIResult;
  要求 ID，インスタンス ID，設定したい値を引き数とする
/* 取得要求関数 */
function CI_get_クラス名_属性名 (
    &request_id: val integer,
    &instance_id: val integer): CIResult;
  要求 ID，インスタンス ID を引き数とする
/* 取得要求継続関数 */
function CI_get_next_クラス名_属性名 (
    &request_id: val integer,
    &instance_id: val integer): CIResult;
  要求 ID，インスタンス ID を引き数とする
```

(2) 使用するデータ

要求 ID: 属性の設定要求や取得要求の識別に使用し，通信スタブが管理する．

トランザクション管理フラグ: 属性の設定/取得状況を表わす WME で，要求 ID，クラスのオブジェクト識別子，属性のオブジェクト識別子，インスタンス ID，MO 操作の実行状況/結果を表わす状態フラグをフィールドを持つ．

通知 ID: トラップを識別するために使われる識別子であり，通信スタブが管理する．

通知フラグ: トラップが通知されたことを示す WME で，通知 ID，トラップ中に含まれる変数の個数，トラップが発行された理由を示す generic-trap，specifi-trap，time-stamp をフィールドに持つ．

通知情報: 通知されたトラップに含まれる変数の内の 1 つの変数を表わす WME．通知 ID，該当トラップの何番目の変数がこの通知情報で表現されているかを示す通知番号，トラップを出した MO のクラスのオブジェクト識別子であるクラス OID，トラップの種類を示すオブジェクト識別子である通知 OID，該当変数の値をフィールドに持つ．

図 3：管理操作関数と使用するデータ

以下，図 4 にユーザプログラムが設定要求を出す時の処理の流れを示す．図中の番号は処理の順序を示す．

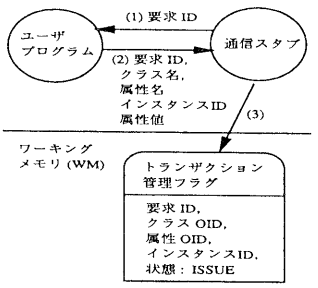

図 4：設定要求処理の流れ

(1) ユーザプログラムは CI_get_id を用いて要求 ID を取得する．

(2) ユーザプログラムは，引き数に要求 ID，インスタンス ID，および属性値を設定し，属性毎に存在する設定要求関数 (**CI_set_クラス名_属性名**) を呼び出す．通信スタブは，対象となる MO を求め，XMP/XOM の mp_get_req を用いて，エージェントに設定要求を行う．

(3) 通信スタブは，OPS83 の WME 生成関数である make を用いて，トランザクション管理フラグを生成する．

図 5 に，ユーザプログラムが出した設定要求に対し，応答が返った時の処理の流れを示す．

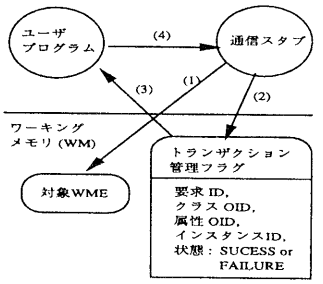

図 5：応答に対する処理の流れ

(1) 設定要求が成功していた場合，通信スタブは設定対象となった属性を持つインスタンスに対応する WME の該当属性の値を OPS83 の WME 更新関数である modify を用いて更新する．

(2) 通信スタブはエージェントの設定結果 (成功，タイムアウト等) に応じてトランザクション管理フラグの状態フラグを modify を用いて更新する．

(3) ユーザプログラムは，パターンマッチングを利用して，トランザクション管理フラグの状態フラグの値から設定要求の結果を知る．

(4) ユーザプログラムが，CI_free_id を呼び出す．通信スタブは，今回使用した要求 ID を再利用可能にし，OPS83 の WME 消去関数である remove を用いて，トランザクション管理フラグを消去する．

設定要求の応答処理同様，取得要求と取得要求継続の応答処理では，通信スタブは取得対象となった属性を持つインスタンスに対応する WME の該当属性の値を更新する．

また，図 6 に，通信スタブがトラップを受信した時の処理の流れを示す．

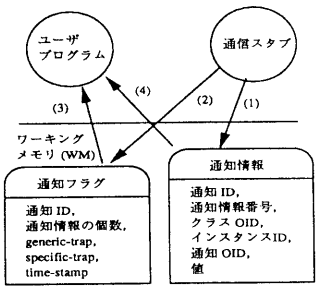

図 6：トラップに対する処理の流れ

(1) 通信スタブが OPS83 の make を用いて通知情報を生成する．

(2) 通信スタブが OPS83 の make を用いて通知フラグを生成する．

(3) ユーザプログラムは，パターンマッチングによって，通知フラグが WM 上に生成されていることを知る．

(4) ユーザプログラムは通知フラグの通知 ID から，同じく WM 中に存在する通知情報を参照し，トラップに含まれていた属性の値を参照する．

上記に示したように，本ツールが生成する管理操作関数は取得した MO の情報を ES のワーキングメモリに自動的に反映する．

4.3 インスタンス管理部の生成

インスタンス管理部は，構成情報に関する以下のデータを外部ファイルに持つ．

エージェント情報: 網管理 ES が対象とするエージェントに関する情報で，エージェント名，コミュニティ名，アドレスを含む．

インスタンス情報: エージェント毎に存在し，インスタンス ID，そのインスタンス属するクラスのオブジェクト ID，該当インスタンスに対応する WME を自動生成する/しないを示す管理対象フラグ等，個々のインスタンスの情報を含む．

インスタンス管理部は，立ち上げ時に，管理対象フラグで指定してある全インスタンスに対し，そのインスタンスが持つ全属性の値を取得し，インスタンスに対応する WME を生成する．ユーザプログラムがエージェント情報とインスタンス情報を書き換えた場合，インスタンス管理部はエージェント情報とインスタンス情報を再度読み込み，管理フラグが「自動生成する」に変わったインスタンスに対し，属性値の取得と WME の生成を行う．

5 評価

GDMO を用いて SNMP の各種 MO をモデル化した NMF (Network Management Forumn) の IIMC (ISO/ CCITT and Internet Management Coexistence) [6] の定義の一部を使用し，生成した通信スタブのプログラム規模と通信速度を評価した．

5.1 生成した通信スタブのプログラム規模

自動生成の対象としたのは，top, internetSystem, interfaces, ifEntry, at, atEntry, ip, ipAddrEntry, ipRouteEntry, ipNetToMediaEntry, icmp の 11 個のクラスである．生成したプログラムの規模を表 2 に示す．同表から，1 クラスあたり生成するソースプログラムのステップ数をセミコロンの数に基づいて求めると，C と OPS83 でそれぞれ約 0.3K と約 1K，計 1.3K ステップとなった．

表 2：生成した通信スタブプログラムの規模

	ソース行数	セミコロン数
固定部分 (C 言語)	9,062	2,653
自動生成部分 (OPS83)	19,384	10,530
自動生成部分 (C)	8,436	3,189

注: 固定部分: 通信環境管理ライブラリ
自動生成部分: 型宣言，管理操作関数，インスタンス管理部

生成した通信スタブによって，網管理プロトコルを意識しないでユーザプログラムを簡潔に記述できる．その例として，(1) sysContact の最新の値を取得するためのルールと，(2) 通信品質が悪いインタフェースを見つけるためのルールを以下に挙げる．

sysContact の値を最新にするためのルール

```
rule Update_syssContact_of_internetSystem
{
---  もしデータが更新されていないならば
  &X (internetSystem);
   ~ (update id = &X.id,
             target = sysContact);
   -->
  local &req_id: integer, &result: CIResult;
```

```
--- 要求 ID を取得し，取得要求用関数を呼び出す
  &req_id = CI_get_id();
  &result =
     CI_Get_internetSystem_sysContact(&req_id,&X.id);
--- 要求が成功したかチェックする
  if (&result <> CI_SUCESS) {
    -- エラー処理
  };
--- データが更新されたことを示すフラグを WM に生成
  make update (id = &X.id; target = sysContact);
};
```

通信品質が悪いインタフェースを見つけるルール

```
rule Check_interface
{
--- もし，上位レイヤに届けられなかったパケットの
--- 数が届けられた数の 0.1%より大きいインタ
--- フェース X が存在し，X のホストが Y であれば
   &X (ifEntry
        (@.ifInErrors+@.ifInDiscards)
        > (@.ifInUcastPkts*.0.001));
   &Y (internetSystem);
   &Z (NameBind sup=&Y.id; sub=&X.id);
   -->
--- Y の管理者に連絡せよと出力する
    write (), |Interface Error: contact |,
      &Y.sysContact, |.|, '\n';
};
```

5.2 生成した通信スタブの処理速度

InternetSystem の属性の 1 つである sysContact に対して，50msec 毎に設定要求を非同期モードで行う OPS83 プログラムを，生成した通信スタブにリンクし，設定要求処理に費やされる通信スタブの処理時間を測定した．取得要求を行うプログラムについても同様の測定を行った．使用した計算機は HP9000/735 であり，HP OpenView が提供する XMP/XOM のライブラリを使用し．エージェントには同一計算機上で動作する snmpd を用いた．

時間の測定は，図 7 に示す T_1 と T_3 の時間を計った．同図中の太線は，処理を行っている期間を示す．

T_1: ユーザプログラムが設定/取得要求の関数を呼び出してから，通信スタブが XMP/XOM の mp_set_req 等を発行し，制御をユーザプログラムに渡すまでの時間
$T2$: エージェントからの応答を待つ時間で，ユーザプログラムが実行可能な時間
T_3: 通信スタブに制御が渡ってから，mp_set_req 等に対する応答を select 関数で検出し，応答処理を行い，ユーザプログラムに制御を渡すまでの時間

測定結果を表 3 に示す．生成した通信スタブの場合，非同期処理ならば，値の設定にかかる時間は 1 属性あたり計 9.4msec，取得にかかる時間が計 5.5msec である．生成した通信スタブは，十分実用的な処理速度を達成していると考えられる．

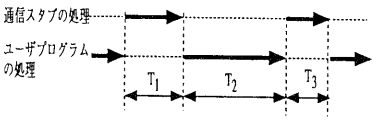

図 7 : 非同期式

表 3 : 処理時間

	T_1	T_3
設定 (50 回の平均)	9.03	0.32
取得 (50 回の平均)	4.26	1.22

6 おわりに

本稿では，網管理 ES から SNMP を利用するための通信スタブを MO の定義から自動生成するツールについて述べた．本ツールが生成する通信スタブは，

(1) MO の種類毎に対応した OPS83 の型宣言

(2) 取得した MO の情報を WM に自動的に反映する管理操作関数

(3) MO と WME の一貫性を管理するためのインスタンス管理部

からなる．

本ツールは属性の種類 (クラス) 毎に 1.3K ステップのプログラムを生成する．これは，ATM 交換機のように MO のクラス数が非常に多くなるエージェントを管理するマネージャのプログラムを開発する際に，生産性を向上させるうえで特に有効となる．また，本ツールが生成する通信スタブプログラムの処理時間は，属性の設定に 9.4ms，取得に 5.5ms であり，実用的な処理速度を達成している．本通信スタブ生成ツールにより，高度な網管理システムの開発が効率的に行えるようになった．

最後に，日頃御指導頂くＫＤＤ研究所浦野所長に感謝します．

参考文献

[1] 宮内他: OSI管理における管理情報データベース (MIB) とその支援系の設計と実現, 信学論, Vol.J-74B-I, No.11, Nov. 1991
[2] 堀内, 小花, 西山, 杉山, 鈴木: OSI 管理のシステム管理機能 (SMF) のためのプログラム開発支援ツールの実装, 信学全大 (93 年春)
[3] Forgy, C.L.: Rete: A Fast Algorithm for Many Pattern/Many Object Pattern Match Problem, Artificial Intelligence, 19(1), 1982
[4] X/Open Company, Ltd.: Draft 3 of X/Open System Management: Management Protocol (XMP) Preliminary Specification
[5] ITU-T Rec. X722, "IT - OSI - Strucure of Management Information: Guidelines for the Definition of Managed Objects,"
[6] NM Forum: Forum 026:Translation of Internet MIBs to ISO/CCITT GDMO MIBs, Octorber 1993