Media Gateway Control Protocol을 위한 프로토콜 스택에 관한 연구

정회원 고 광 만*

A Study on the Protocol Stack for the Media Gateway Control Protocol

Kwang-Man Ko* *Regular Member*

요 약

현재 국내·외적으로 VoIP 관련 시장의 지속적인 성장으로 인해 VoIP 표준 프로토콜에 대한 국제 표준화 작업과 병행하여 프로토콜 스택과 같은 소프트웨어 개발이 절실한 상황이다.

본 논문에서는 IETF RFC2705에서 제시한 MGCP 문법을 기반으로 문법 지식적 변환 기법으로 Media Gateway가 전달하는 MGCP 요청 메시지를 구조체 형태로 변환하는 앤코더와 Media Gateway Controller로부터 전달되는 구조체 형태의 응답 메시지를 텍스트 형태로 변환하는 디코더를 구현하였다. 이를 위해 BNF 형식의 MGCP 문법을 기술하여 구문 분석 동작 후 구문 트리 및 MGCP 문법으로부터 생성 가능한 모든 정보를 저장할 수 있는 구조체 형태의 헤더 파일을 자동 생성하였다.

ABSTRACT

Accordingly it is very important to not only develop the stack of protocol, but also try an international standardization regarding the standard protocol of VoIP. Has compared to the advanced countries having already some success in commercialization, Korea is relatively much less involved in relation to this technology and endeavors. So far nothing is unfortunately there any attempt to try any research with respect to the development of the protocol stack relating to such control of gateway as MGCP, MEGACO, SIP, SDP. The reasons come from the low level of infrastructue, the shortage of the time and technology required at the moment, and so on.

In this regards, this paper is focused on developing a protocol stack made with encoder/decoder, the generator of the header file, syntax analyzer etc. based on the protocol grammars of Media Gateway Control Protocol supported by ITU-T or IETF2705. For the sake of it, we develops the API for encoding/decoding as applying the method of syntax-directed to each protocol grammar and produces header file generator automatically.

Ⅰ. 서 론

최근 서비스 질(Quality of Service; QoS)과 비용 면에서 장점을 갖는 데이터 네트워크를 통해 음성 신호의 전달하고자 하는 VoIP(Voice over IP)에 관련된 기술 개발 및 연구가 국내외적으로 활성화되고 있다. 초창기 IP 네트워크는 데이터 송수신을 목적으로 만들어진 네트워크로 아날로그 신호인 음성을 전달하도록 고안되지는 않았지만 IP 네트워크의 월등히 저렴한 비용 때문에 IP 네트워크를 통해 음성을 주고받을 수 있다면 값비싼 국제 전화료 또는 장거리 전화료를 파격적으로 줄일 수 있는 점에 착안해 꾸준히 IP 네트워크상에서 음성을 전달하려는 기술의 진보가 있어 왔다.[9]

* 광주여자대학교 정보통신부
논문번호 : 00417-1025, 접수일자 : 2000년 10월 25일

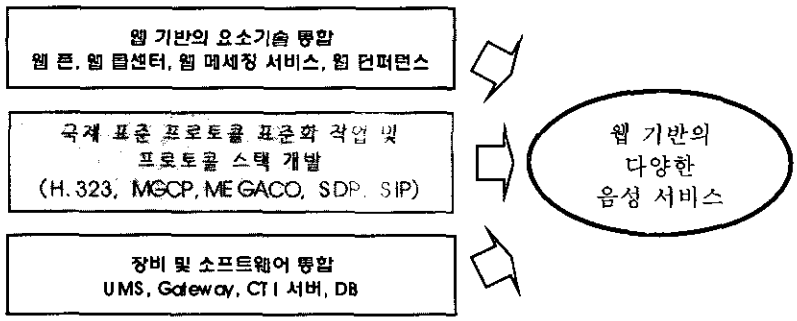

그림 1. VoIP 시장 및 연구 동향

1995에 VoIP 포럼이 인터넷상에서 음성 신호를 전송하는 전화 제품들과 사설 IP 기업 네트워크간의 상호 호환을 가능하게 하고 이러한 기술을 개선하고자 미국에서 시작되었다. 그 후 세계의 많은 기업들과 연구진의 참여로 IP 네트워크상에서 비디오, 오디오, 데이터 전송을 위한 표준 프로토콜인 H.323 등을 정의하였다. 국내에서는 한국통신을 비롯해 많은 통신 관련 업체들이 VoIP를 이용한 인터넷폰이 점차 실용화 단계에 접어들고 있다. 특히, 인터넷폰에 필수적인 장비인 게이트웨이 장비를 출시하는 업체들이 속속 등장하고 있다. 이러한 VoIP 관련 시장 및 연구 동향은 국내외적으로 그림 1과 같은 분야에 집중되고 있다.

특히, 인터넷 표준 프로토콜 및 VoIP 표준 프로토콜은 IETF(The Internet Engineering Task Force) 커뮤니티 등을 중심으로 H.323 중심의 ITU-T 표준안 업그레이드, 게이트웨이 제어 관련 프로토콜 국제 표준화 및 프로토콜 스택 개발, RSVP/DiffServ 등 품질(QoS) 보장 기술 적용 방안 반영, H.235 등 보안 기능 강화, Mobile H.323 등 차세대 통신망과의 연동 방안 표준화 작업과 같은 부분을 꾸준히 추진되어오고 있다.[6][7][8][9]

현재 국내외적으로 VoIP 관련 시장의 지속적인 성장으로 인해 VoIP 표준 프로토콜에 대한 표준화 작업과 병행하여 프로토콜 스택의 개발이 절실한 상황이다. 국외에서 게이트웨이를 제어하기 위한 VoIP 표준 프로토콜에 대한 프로토콜 스택의 개발이 진행중이며 일부 제품은 상용화되어 판매되고 있지만 아직은 초기 단계이다. 실제 상용화되어 판매되고 있는 제품은 엄청난 비용을 지불해야 하는 어려움을 가지고 있다. 현재까지 국내에서 MGCP, MEGACO, SIP, SDP 등 게이트웨이 제어 관련 프로토콜 스택의 개발을 위한 연구는 아직까지 열악한 환경과 많은 시간 및 기술을 요하는 관계로 인해 이를 진행하기 위한 시도만 일부 이루어지고 있다.

본 논문에서는 IETF RFC2705[1]서 제시한 MGCP 문법을 기반으로 문법 지식적 변환 기법으로 Media Gateway가 전달하는 MGCP 요청 메시지를 구조체 형태로 변환하는 엔코더와 Media Gateway Controller로부터 전달되는 구조체 형태의 응답 메시지를 텍스트 형태로 변환하는 디코더를 구현하였다. 이를 위해 BNF 형식의 MGCP 문법을 기술하여 구문 분석 동작 후 구문 트리 및 MGCP 문법으로부터 생성 가능한 모든 정보를 저장할 수 있는 구조체 형태의 헤더 파일을 자동 생성하였다.

Ⅱ. MGCP 특징

2.1 Media Gateway Control Protocol

인터넷을 통해 화상 전화나 음악 서비스를 손쉽게 구현하는 인터넷 동영상 프로토콜인 MGCP가 기존의 인터넷 전화 기술 표준인 "H.323"을 제치고 차세대 인터넷 텔레포니(telephony) 시스템의 핵심 기술로 급부상하고 있다. MGCP가 상용화되면 인터넷과 전화를 연결하는 시스템인 인터넷 텔레포니를 통해 화상 전화나 오디오 등과 같은 음성 서비스 기능을 손쉽게 제공할 수 있게 된다. 특히 기존 H.323제품이 컴퓨터 단말기를 기반으로 하는 데 반해 MGCP는 대용량 서버용으로 확장성과 다양한 부가 기능을 구현할 수 있는 특징을 가지고 있다.[1]

MGCP는 MGC(Media Gateway Control) 또는 호출 에이전트(call agents)라 불리우는 외부 호출 제어로부터 텔레포니 게이트웨이를 제어하기 위해 제시된 표준 프로토콜이다. 이러한 MGCP 프로토콜

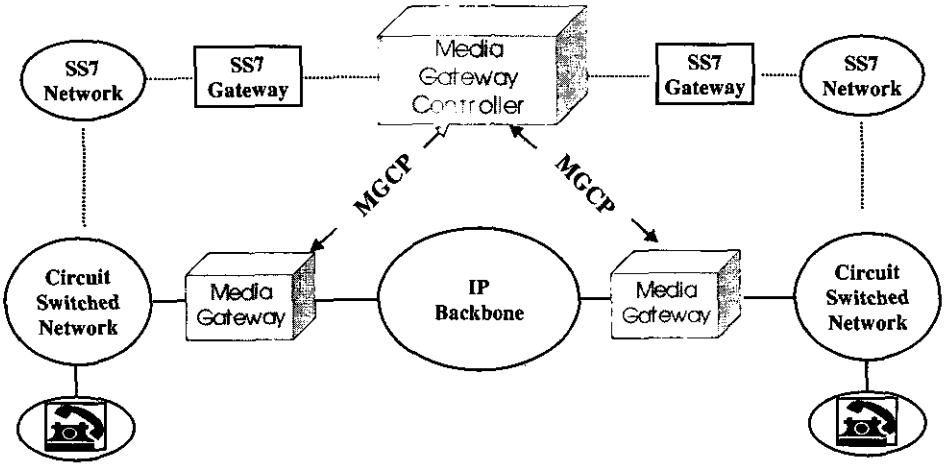

그림 2. Media Gateway, Media Gateway Controller, MGCP

은 IETF의 RFC에 의해 제안된 것으로 VoIP 솔루션의 핵심 기술로서 그림 2와 같이 미디어 게이트웨이(MG)와 미디어 게이트웨이 제어기(MGC) 사이에서 사용된다.

2.2 MGCP 명령어 : IETF RFC2705

게이트웨이를 제어하기 위한 MGCP 명령어는 연결(connection)을 처리하는 부분과 엔드포인트(endpoint)를 처리하기 위한 10개의 명령어로 구성되어 있다. 모든 명령어는 명령어 헤더 부분과 선택적으로 기술되는 세션 부분으로 구성되어 있으며 응답 메시지도 헤더 부분과 선택적으로 기술되는 세션 부분으로 구성되어 있다.[1]

"EndPointConfiguration" 명령어는 호출 에이전트가 게이트웨이 발생하는 명령어로서 엔드포인트에 수신되는 신호의 엔코딩을 지정하는데 사용되며 다음과 같은 형태를 가지고 있다.

ReturnCode <- EndPointConfiguration(EndpointId, BearerInformation)

"EndpointId"는 게이트웨이에 존재하는 엔드포인트의 이름을 지칭하는데 사용되지만 "all of" 와일드 문자를 사용할 수 있다. "BearerInformation" 수신된 데이터의 코딩을 정의하는 매개변수를 나타낸다. "NotifyRequest" 명령어는 호출 에이전트가 게이트웨이 발생하는 명령어로서 엔드포인트에서 발생되는 특정 이벤트에 대한 발생을 호출 에이전트에게 통보하도록 게이트웨이에 요청하는데 사용된다. "Notify" 명령어는 게이트웨이가 호출 에이전트에게 요청된 이벤트 발생을 통보하는데 사용된다. "CreateConnection" 명령어는 게이트웨이 안에 존재하는 엔드포인트와 연결을 설정하기 위해 사용된다. "ModifyConnection"는 호출 에이전트가 기존에 설정된 연결에 관련있는 매개변수 등을 변경하는데 사용되는 명령어이다. 이 밖에도 "Delete Connection", "AuditEndPoint", "AuditConnection", "RestartInProgress", "MoveConnection" 등의 명령어 등이 게이트웨이를 제어하기 위한 명령어로서 사용된다. 이러한 명령어들은 "EndPointConfiguration"과 같이 자신의 고유한 기본적인 매개변수 및 선택적인 매개변수를 가지며 해당 명령어를 수행한 후 반환 코드를 생성한다. MGCP 명령어는 RFC2705에 정의된 각 명령어의 해당되는 매개변수를 활용하여 다양한 요청(request) 동작 및 응답(response) 동작을 수행한다.

각 명령어가 갖는 매개변수는 명령어의 종류에 따라 서로 다른 매개변수를 갖을 수 있으며 디폴트(default), 선택(option), 금지(forbidden) 형태로 구분된다. 연결 매개변수의 경우 PS, OS, PR, OR, PL, JI, LA 등과 같은 매개변수를 갖으며 문자열 형식으로 해당되는 값과 함께 엔코딩되며 하나 이상의 매개변수에 대해서는 콤마로서 구분된다. 연결 명령어에 관련된 매개변수 기술 예는 그림 3과 같다.

P: PS=1245, OS=62345, PR=0, OR=0, PL=0, JI=0, LA=48

그림 3. 연결 명령어의 매개변수 기술 예

그림 4에서는 "CreateConnection" 명령어의 요청 메시지와 응답 메시지 및 세션을 RFC2705에 정의된 Augmented-BNF 형식을 기반을 작성한 예이다.

```
CRCX 1204 aaln/1@rgw-2567 .whatever.net MGCP 1.0
NCS 1.0
        C: A3C47F21456789F0
        L: p:10, a:PCMU
        M: recvonly

        200 1204 OK
        I: FDE234C8
```

그림 4. "CreateConnection" 요청 메시지와 응답 메시지 예

Ⅲ. MGCP 프로토콜 스택 구현

3.1 프로토콜 스택 구현 개요

본 논문에서는 IETF RFC2705에서 제시한 MGCP 문법을 기반으로 문법 지식적 변환 기법으로 MGCP 요청 메시지를 구조화된 형태로 변환하는 엔코더와 구조화된 형태의 응답 메시지를 텍스트 형태로 변환하는 디코더를 구현한다. 이를 위해 BNF 형식의 MGCP 문법을 기술하여 구문 분석 동작후 구문 트리 및 헤더 파일을 정형화된 방법으로 생성할 수 있는 프로토콜 스택을 그림 5와 같이 시스템 구성 모델을 설정하고 관련 부분을 구현하였다.

프로토콜을 위해 MGCP 문법은 bison 파서 생성기에 입력되는 형태로 정형화하여 BNF 형식으로 재구성되었으며 문법 기술시에 구문 트리 생성 및 헤더 파일 생성을 위한 의미 수행 코드를 기술한다. 따라서 파서 생성기는 정형화 방법으로 기술된 MGCP에 대한 BNF 문법을 입력으로 받아 어휘 분석기, 구문 분석기, 헤더 파일 생성기 등을 자동으로 생성한다. 또한 입력되는 MGCP 텍스트 메시지에 대해 파서는 구문 분석을 수행한 후 구문 트리를 자동으로 생성하며 MGCP 텍스트 메시지에 대한 정보를 저장하기 위해 헤더 파일을 생성한다. 헤더 파일 생성기에 의해 자동으로 생성되는 헤더 파일은 문법에서 허용되는 모든 가능한 규칙을 포함하고 있으므로 모든 MGCP 메시지 정보를 저장할 수 있다. 따라서 헤더 파일은 입력으로 허용되는 요청 및 응답 메시지에 대한 모든 정보를 저장하고 있는 저장소 역할을 한다. 메시지 엔코더(message encoder)는 입력으로 허용되는 올바른 텍스트 MGCP 메시지에 대해 헤더 파일이 저장된 정보를 참조하여 구조체(struct) 형식의 구조화된 메시지를 출력한다. 또한 메시지 디코더(decoder)는 헤더 파일구조에 저장된 구조체 형식의 메시지 정보를 다시 문법 규칙을 적용하여 텍스트 MGCP 메시지 형태로 반환하는 역할을 한다.

3.2 어휘 분석 및 구문 분석

어휘 분석기 및 구문 분석기는 IETF RFC2705[1]에서 제안한 MGCP 문법을 기반으로 기술한 형태를 입력으로 받아 MGCP 메시지에 대한 어휘 분석 및 구문 분석을 수행한다. 이를 위해 본 연구에서는 컴파일러 전단부 구성의 자동화 도구인 flex[3]를 이

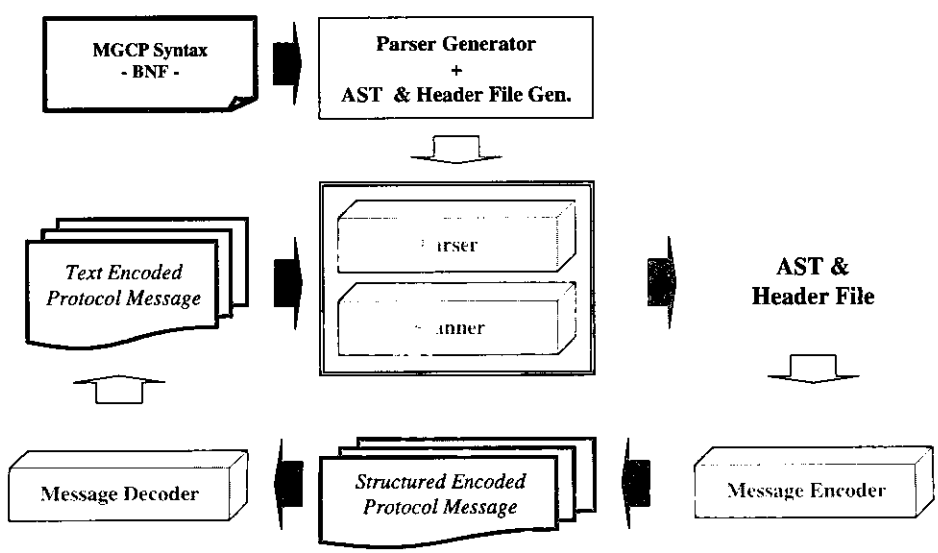

그림 5. MGCP 프로토콜 스택 구현 모델

용하여 어휘 분석기를 자동 생성하였으며 bison[4]을 이용하여 구문 분석기를 자동 생성하였다.

어휘 분석기와 구문 분석기는 MGCP 요청(request) 메시지에 대해 어휘 분석 및 구문 분석 동작을 수행한 후 올바른 텍스트 메시지에 대해 본 연구에서 설계한 구문 트리(Abstract Syntax Tree) 및 헤더 파일을 생성한다. 구문 트리는 파서 생성기의 출력인 구문 분석기에 입력되는 MGCP 메시지에 대한 중간 표현 형태로서 올바른 구조를 가진 메시지에 대한 정보를 저장하고 있으며 이러한 정보를 이용하여 헤더 파일이 정의된 구조화된 형태의 메시지로 변환된다. 헤더 파일은 구문 트리에 저장된 MGCP 요청 메시지 정보를 구조체 형식으로 변환하여 저장하며 Media Gateway로부터 발생되는 응답(reply) 메시지에 대한 정보를 저장한 후 다시 텍스트 메시지로 변환하기 위한 정보를 저장하는 곳으로 사용된다.

텍스트 형태의 MGCP 요청 메시지에 대한 구문 분석을 수행한 후 구문 트리 및 헤더 파일을 생성하기 위해 IETF RFC2705의 Augmented-BNF 문법을 기반으로 하여 파서 생성기인 BISON을 입력을 그림 6과 같이 BNF 형식으로 기술하였다.

3.3 구문 트리 및 헤더 파일

올바른 MGCP 텍스트 메시지 입력에 대해 어휘 분석 및 구문 분석을 수행한 후 구조체 형식을 가진 엔코딩된 메시지를 생성하기 위해 입력 메시지에 대한 그림 7과 같은 노드 구조를 갖는 구문 트리를 생성한다.

```
typedef enum kind { terminal, nonterminal } n_kind ;
typedef struct nodetype {
        n_kind node_kind ;  //노드 종류
        char  *node_name ; //노드 이름
        struct nodetype *brother ;
        struct nodetype *son ;
} NODE ;
```

그림 7. 구문 트리 노드 구조

구문 트리 생성에 필요한 노드 구조을 이용하여 본 연구에서 기술한 문법 생성 규칙에 따라 노드를 구성하고 궁극적으로 구문 트리를 생성하기 위해 BISON 입력으로 BNF 형식의 MGCP 문법과 해당 문법이 축약(reduce) 또는 이동(shift) 되었을 때 구문 트리를 구성하는 동작 코드를 작성하였다. 먼저 노드를 구성하기 위해서는 문법 규칙에서 이동이 발생하는 경우 그림 8에 기술된 buileNode() 함수를 호출하여 노드를 구성한다.

```
MGCPMessage
        : MGCPCommand      { $$ = maketree(NULL, $1, NIL) ; }
        | MGCPResponse     { $$ = maketree(NULL, $1, NIL) ; }
        ;
MGCPCommand
        : MGCPCommandLine  TEOL  MGCP_para_list TEOL
          { $$ = maketree(NULL, $1, $2, $3, $4, NIL) ; }
        | MGCPCommandLine  TEOL  MGCP_para_list
          { $$ = maketree(NULL, $1, $2, $3, NIL) ; }
        ;
MGCPCommandLine
        : MGCPVerb WSP_list Tran_ID WSP_list endpointName WSP_list
MGCPversion
          { nt1 = MakeNode( nonterminal, Buffer ) ;
            $$ = maketree(NULL, $1, $2, nt1, $3, $4, $5, $6, $7, NIL) ;
          }
        ;
MGCPVerb
        : TEPCF   { $$ = maketree(NULL, $1, NIL) ; }
        | TCRCX   { $$ = maketree(NULL, $1, NIL) ; }
        | TMDCX   { $$ = maketree(NULL, $1, NIL) ; }
        | . . .
        | extensionVerb        { $$ = maketree(NULL, $1, NIL) ; }
        ;
        // 생략
```

그림 6. MGCP 메시지 BISON 입력 문법 예

```
NODE *buildnode(struct tokentype token) {
        NODE *ptr;
        ptr = (NODE *) malloc (sizeof(NODE));
        if (!ptr) { printf(" buildnode malloc error \n ");
            exit(1);
        }
        ptr->token = token;
        ptr->noderep = terminal;
        ptr->son = NULL;
        ptr->brother = NULL;
        return(ptr);
}
```

그림 8. 노드 구성 함수 : buildNode()

하나의 생성 규칙이 축약되면 그림 9와 같은 buildTree() 함수를 호출하여 구문 트리를 완성한다.

실질적으로 MGCP 요청 메시지에 대한 구문 트리 생성 결과는 그림 10과 같은 구조를 갖는 구문 트리를 생성한다.

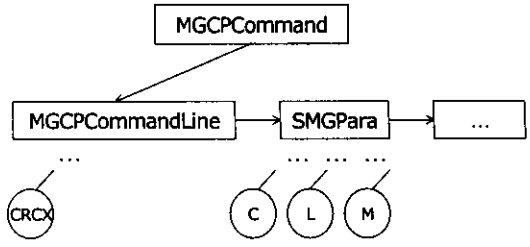

그림 10. 구문 트리 생성 예

3.4 엔코더 및 디코더

헤더 파일은 MGCP 문법으로부터 생성 가능한 모든 규칙에 대한 정보를 저장하기 위해 사용되는 그림 11과 같은 구조체로서 구문 분석기에 의해 자동 생성된다. 이를 위해 본 연구에서는 문법 기술시에 각 문법 규칙으로부터 생성될 수 있는 모든 가능한 규칙에 대한 정보를 저장할 수 있도록 구문 트리 생성 정보를 참조하여 의미 수행 코드를 작성하였다. 따라서 헤더 파일은 MGCP 요청 메시지를

```
NODE *buildtree(int nodenum, int rhslen)
{
        int i, j, start;
        NODE *n, *first, *temp;
        i =  yyTop - rhslen + 1;
        while (i <= yyTop && valueStack[i] == NULL) i++;
        if (!nodenum && I > yyTop) retrun NULL;
        start = i;
        while (i <= yyTop-1) {
                j = i + 1 ;
                while ((j <= yyTop && valueStack[j] == NULL)) j++ ;
                if (j <= yyTop) {
                temp = valueStack[i] ;
                while (temp->brother) temp = temp->brother ;
                temp->brother = valueStack[j] ;
                }
                i = j;
        }
        first = valueStack[start];
        if (nodenum) {
                temp = (NODE *) malloc(sizeof(NODE));
                if (!temp) { printf(" buildtree malloc error\n");
                    exit(1);
                }
        temp->token.tokennumber = nodenum;
        temp->token.tokenvalue  = NULL;
        temp->noderep = nonterm;
        temp->son = first; n->brother = NULL;
        return(temp);
        }
        else {
                if (i > 0) return NULL;
    else
        return first ;
} /* end buildtree */
```

그림 9. 구문 트리 구성 함수 : buildTree()

구조체 형식으로 변환하여 저장하기 위한 저장소로서 사용되며 또한 구조체에 저장된 응답(reply) 메시지를 텍스트 메시지로 변환하여 사용자에게 전달하기 위한 정보를 저장하는 곳으로 사용된다.

입력되는 모든 요청 메시지는 구문 트리를 참조하여 인코더에 의해 구조체 형태로 변환되며 Media Gateway Controller로부터 전달되는 응답 메시지도 헤더 파일에 저장된 후 디코더에 의해 사용자에게 텍스트 메시지로 전달된다. 따라서 엔코더는 MGCP 문법으로부터 생성되는 헤더 파일에 텍스트 메시지를 구조체 형식에 맞게 변환하여 저장하는 기능을 하며 디코더는 구조체에 저장되는 응답 메시지를 다시 텍스트 메시지로 변환하는 역할을 한다.

실질적으로 적용된 입력에 대한 검증을 위해 다음과 같은 MGCP 메시지에 대한 구문 트리의 구성은 최종적으로 12와 같은 터미널로 구성된다.

구문 분석 과정을 통해 생성된 요청 메시지의 터미널 생성 과정은 문법에 생성할 수 가능성의 일부

입력(요청) 메시지 : RQNT 1201 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0

[T] RQNT
[T] 1201
[T] aaln
[T] /
[T] 1
[T] @
[T] rgw-2567.whatever.net
[T] MGCP
[T] 1.0
[T] NCS
[T] 1.0

그림 12. 구문 트리 생성 예

에 해당하므로 각각의 터미널을 헤더 파일 생성기에 의해 생성된 구조체에 저장하는 것은 문법을 참조하여 구성하였다.

입력에 대한 MGCP 응답 메시지의 최종 출력 결과는 헤더 파일에 저장된 메시지의 내용을 역 추적하여 다시 텍스트 메시지로 형태로 변환하는 과정을 통해 확인하였다. 이를 좀더 실질적으로 검사하

```
typedef struct MGCPCommand {
        unsigned short  choice;
        #define     rqEPCF_chosen 1
        #define     rqCRCX_chosen 2
        #define     rqMDCX_chosen 3
        #define     rqDLCX_chosen 4
        #define     rqRQNT_chosen 5
        #define     rqNFTY_chosen 6
        #define     rqAUEP_chosen 7
        #define     rqAUCX_chosen 8
        #define     rqRSIP_chosen 9
        #define     rqMVCX_chosen 10
        union {
                EndPointConfigurationCommand    rqEPCF;
                CreateConnectionCommand         rqCRCX;
                ModifyConnectionCommand         rqMDCX;
                DeleteConnectionCommand         rqDLCX;
                NotificationRequestCommand      rqRQNT;
                NotifyCommand                           rqNFTY;
                AuditEndPointCommand                    rqAUEP;
                AuditConnectionCommand          rqAUCX;
                RestartInProgressCommand        rqRSIP;
                MoveConnectionCommand           rqMVCX;
    } u;
} MGCPCommand;

typedef struct MGCPRequest {
        unsigned char                   bit_mask;
        MGCPCommandHead  mgcpCommandHead;
        MGCPCommand                     mgcpCommand;
} MGCPRequest;
```

그림 11. 헤더 파일의 구조

기 위해 입력되는 요청 메시지를 엔코더에 의해 구조체로 저장한 후 다시 구조체에 저장된 정보를 읽어 처음과 일치하는 메시지를 생성하는 결과를 통해 본 시스템의 타당성을 검증하였다.

Ⅳ. 결론 및 향후 연구과제

현재 국내외적으로 VoIP 관련 시장의 지속적인 성장으로 인해 VoIP 표준 프로토콜에 대한 표준화 작업과 병행하여 프로토콜 스택의 개발이 절실한 상황이다. 국외에서 게이트웨이를 제어하기 위한 VoIP 표준 프로토콜에 대한 프로토콜 스택의 개발이 진행중이며 일부 제품은 상용화되어 판매되고 있지만 아직은 초기 단계이다. 실제 상용화되어 판매되고 있는 제품은 엄청난 비용을 지불해야 하는 어려움을 가지고 있다. 현재까지 국내에서 MGCP, MEGACO, SIP, SDP 등 게이트웨이 제어 관련 프로토콜 스택의 개발을 위한 연구는 아직은 열악한 환경과 많은 시간 및 기술을 요하는 관계로 인해 이를 진행하기 위한 시도만 일부 이루어지고 있다.

본 논문에서는 IETF RFC2705에서 제시한 MGCP 문법을 기반으로 문법 지식적 변환 기법으로 Media Gateway가 전달하는 MGCP 요청 메시지를 구조체 형태로 변환하는 엔코더와 Media Gateway Controller로부터 전달되는 구조체 형태의 응답 메시지를 텍스트 형태로 변환하는 디코더를 구현하였다. 이를 위해 BNF 형식의 MGCP 문법을 기술하여 구문 분석 동작 후 구문 트리 및 MGCP 문법으로부터 생성 가능한 모든 정보를 저장할 수 있는 구조체 형태의 헤더 파일을 자동 생성하였다.

본 논문에서 시도한 문법 지시적 변환 방법을 이용하여 VoIP 표준 프로토콜 문법에 대한 프로토콜 스택에 관한 연구는 현재까지 MGCP에 국한되어 시도되었지만 앞으로 SDP, MEGACO, SIP와 같은 프로토콜에서도 본 연구에서 제안한 모델을 기반으로 구현할 예정이다. 또한 본 연구 결과에 대한 성능 평가 방법를 현재까지는 고려하지 않고 단지 모델에 대한 구현으로 국한되어 있지만 향후에 다른 연구 결과와 비교 분석을 진행하여 성능 개선을 보완할 예정이다.

참 고 문 헌

[1] IETF, rfc2705, Media Gateway Control Protocol(MGCP) Version 1.0

[2] ITU-T, APC-1855, Proposal For an Advanced Audio Server Package for H.248

[3] Bill Douskalis, "IP Telephony," pp7-12, Prentice-Hall, Inc. 2000.

[4] M. Marjalakso, "Security requirements and Constraints of VoIP," HUT. http://www.tml.hut.fi.

[5] FLEX : http://www.combo.org/lex_yacc_page/

[6] BISON : http://www.combo.org/lex_yacc_page/

[7] http://www.netmanias.com

[8] http://www.hsswrold.com

[9] http://www.prptocols.com

[10] http://www.catapult.com

[11] http://www.ietf.com

[12] http://www.iab.org/iab/

[13] http://www.iana.org/

[14] http://www.irtf.org/

고 광 만(Kwang-Man Ko)

1991년 2월 : 원광대학교 컴퓨터공학과 졸업

1993년 2월 : 동국대학교 컴퓨터공학과 석사학위 취득.

1998년 2월 : 동국대학교 컴퓨터공학과 박사학위 취득.

1998년 3월~현재 : 광주여자대학교 컴퓨터학부 전임강사.

<주관심 분야> 프로그래밍 언어론, 컴파일러 구성론.