

DRAFT

*VoIP Forum Technical Committee
Contribution*

Call Management Agent System Specification

Subject: Call Management Agents System Specification

Contact: Opher Kahane
VocalTec. LTD
1 Maskit St., Herzeliya, Israel
Phone: 972-9-525823
Fax: 972-9-561867
E-mail: odk@vocaltec.com

Date: Aug. 14th

Location: Chicago

Distribution: Participants in the Voice over IP Forum Technical Committee in Chicago August 14, 1996.

Abstract: The following document presents the Call Management Agents System architecture and protocol.

Status: To be presented to the VoIP meeting After review and assuming approval from the VoIP committee this work will next be made public as an Internet Draft. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas and its Working Groups.

Notice: This contribution is presented by VocalTec to the Voice over IP Forum as a basis for discussion. This should not be construed as a binding proposal on VocalTec. Specifically, VocalTec reserves the right to request amendments and/or modifications to the proposal in the future.

Authors:

<i>Opher Kahane</i>	<i>VocalTec Ltd.</i>
<i>Dror Tirosh</i>	<i>VocalTec Ltd.</i>
<i>Dave Oran</i>	<i>Cisco Inc.</i>
<i>Ken Krechmer</i>	<i>Action Consulting</i>

*File:x:\tech\servers\Cmaspec
Created by: ODK
Created on: 08/08/96
Version: 11, 08/15/96 9:53 AM
Printed on: 4/16/2007 3:16 PM
Updated by: odk*

1. INTRODUCTION	4
1.1 PURPOSE OF DOCUMENT	4
1.2 OVERVIEW	4
1.3 SYSTEM OPERATING ENVIRONMENT	6
1.4 TERMINOLOGY	7
1.5 REFERENCES.....	10
2. SYSTEM ARCHITECTURE.....	11
2.1 OVERVIEW	11
2.2 CMA	11
2.3 CMAA	13
2.4 CMAS	13
2.5 CMAC.....	13
2.6 CMAP	14
3. SECURITY	15
3.1 AUTHENTICATION.....	15
3.2 DIGITAL SIGNATURES	15
3.3 TOKEN BASED ACCESS CONTROL	15
4. EXTERNAL INTERFACES	16
4.1 DIRECTORY SERVICES: LDAP, X.500	16
4.2 H.323 GATEKEEPER.....	16
4.3 VoIP/H.323 GATEWAY	17
5. CMA SEMANTICS.....	18
5.1 CMA OBJECT INSTANCE DATA	18
5.1.1 CMAA.....	18
5.1.2 Target CMA.....	18
5.1.3 CTSPEC	18
5.2 CALL MANAGEMENT AGENT METHOD SUMMARY	21
5.3 METHODS FOR CREATING AND MAINTAINING CMAS	21
5.3.1 CreateCMA.....	21
5.3.2 RemoveCMA	22
5.3.3 AddSynonym	22
5.3.4 RemoveSynonym	22
5.3.5 SetTargetCMAA.....	23
5.3.6 GetTargetCMAA	23
5.3.7 UnsetTargetCMAA.....	24
5.4 METHODS FOR MAINTAINING CTSPECS	24
5.4.1 SetCTSpecifier	24
5.4.2 KeepAlive	25
5.5 METHODS FOR OBTAINING INFORMATION FROM A CMA	25
5.5.1 Resolve	25
5.5.2 GetSynonyms	26
6. PROTOCOL SYNTAX.....	27
6.1 INTRODUCTION.....	27
6.2 PROTOCOL TRANSPORT.....	27
6.3 GENERAL PROTOCOL NOTES	28
6.4 CONNECT SEQUENCE.....	28
6.4.1 Overview.....	28
6.4.2 PDU structure.....	28

6.5 BASIC CMAP OPERATIONS.....	30
6.5.1 Overview.....	30
6.5.2 PDU Structure	31
6.6 CMA SPECIFIC PROFILE.....	36

1 . I N T R O D U C T I O N

1.1 Purpose Of Document

This document describes an agent based call management system and protocol for real time audio communication over IP networks.

Client pc and workstation software has emerged in the marketplace from multiple vendors to support voice over the Internet. Such existing software does not allow interworking between vendors as voice coding, silence compression, addressing and related functions are not compatible. The purpose of this document is to develop a common specification that will allow vendors who support this specification to offer interworking products in terms of the call management aspects. The client to client presentation, session and transport layers are defined in separate VoIP specifications.

It is the goal of this effort to support point-to-point voice and similar audio communications over IP networks in a manner similar and compatible (via gateways) with existing SCN telephone calls. Every attempt has been made to utilize existing IETF protocols and services and ITU protocols. While only point-to-point service is supported in this document every attempt has been made to allow extensibility such that multipoint communications may be implemented in the future in a manner backward compatible with this document and existing formal multi-point conferencing standards.

1.2 Overview

The call management agent system provides intelligent, communication terminal independent call management services. It is an essential link in providing the call setup information for Internet based telecom services including IP to IP, SCN to IP, IP to SCN, and SCN to SCN calls. This includes managing the various communication terminal addresses¹ of a given person or organization, the ability to provide the various dynamic mappings between addresses to allow all combinations of calls, and the ability to intelligently route calls according to agent based logic. The following scenario with the accompanying diagram portray the underlying concept.

Suppose Jack has three communication terminals - a home phone, a business phone and an Internet phone. During working hours, he wants all calls to be routed to his business phone. When he's at home, he wants all calls to be routed to his home phone, but calls from Joe to be routed to his laptop based Internet Phone. Alas, Jack has only a dialup account at an ISP. This means he doesn't have a fixed IP and it might be the case that he isn't online at a given time. In this case, he wants Joe's calls to be routed to his home phone number. Jack would "inject" this logic (0), along with the list of communication terminals he supports into his CMA, so the CMA will be able to perform the call routing accordingly. Now, Jack can be accessed by these various communication through a single logical address, his CMAA.

¹ This includes the resolving *dynamic IP addresses* which are typically used by dialup Internet subscribers.

Now suppose Joe, an Internet Phone user, wants to call Jack. Suppose Joe doesn't have Jack's CMAA. He would thus first consult a white pages directory service, to get the CMAA (1). Now his Internet Phone would contact his own CMA and request it to contact Jack (2). Joe's CMA will now locate Jack's CMA, and contact it, identifying itself as Joe's CMA (3). Jack's CMA will now perform the computation according to the logic previously injected into it. If Jack's online, it will reply with the *current* IP address of Jack's Internet Phone. Otherwise, it will provide Joe's CMA with the E.164 telephone number which will be resolved to the appropriate IP address of the telephony gateway to call out from.

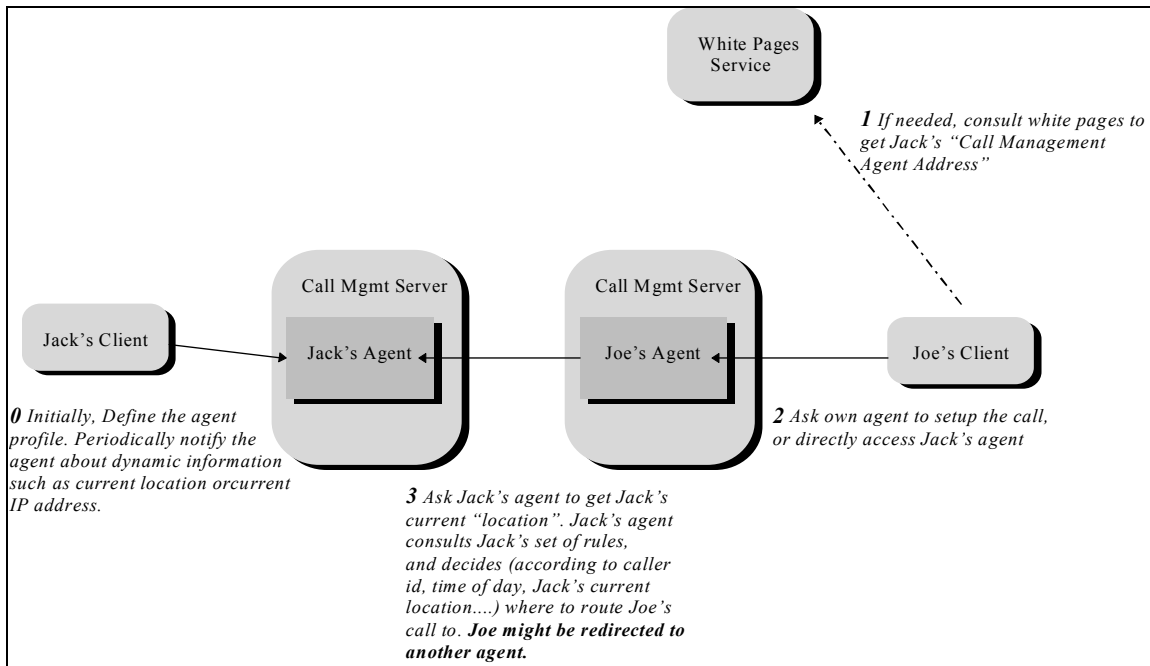


Figure 1-1 CMA System Sample Scenario

Given the needed call setup information, Joe's Internet Phone can now call the remote terminal (Be it a telephony gateway which in turn will call Jack's home E.164 number, or Jack's actual Internet Phone address) directly.

1.3 System Operating Environment

The initial implementation of CMA system supports point-to-point communications. Interworking via H.323 gateways to SCN (Switched Circuit Network) communications equipment is provided. This operating environment is termed the VOIP Environment.

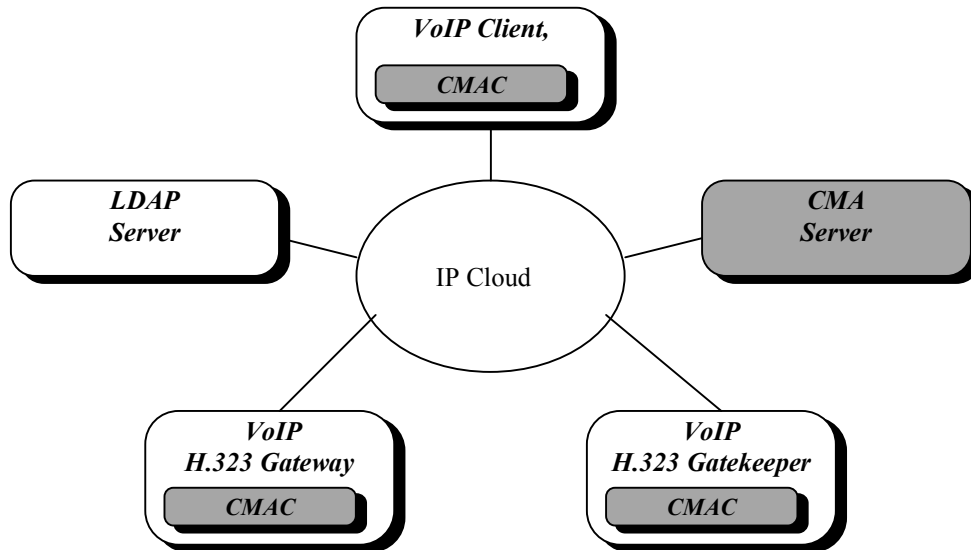


Figure 1-2 The Operating Environment

1.4 Terminology

Addressable: An entity on the Internet having a Transport Address. Not the same as being callable. A client or server is addressable and callable. A gatekeeper is addressable but not callable.

Call: Point-to-point multimedia communication between two Internet endpoints. The call begins with the call setup procedure and ends with the call termination procedure. The call consists of the collection of reliable and unreliable channels between the endpoints. In case of interworking with some SCN endpoints via a gateway, all the channels terminate at the Gateway where they are converted to the appropriate representation for the SCN end system.

CMA: Call Management Agent

CMA: Call Management Agent Address. An address through which a given CMA can be located and then accessed.

CMAS: Call Management Agent Server

CMA Sys: Call Management System

CMA Sys Entity: Any CMA Sys component, including client(s) and server(s).

CMAC: Call Management Agent Client

CMAP: Call Management Agent Protocol. The protocol between a CMAC and a CMAS.

CMA Logic: The computation performed by the CMA for a specific request.

Communication Terminal (CT): A terminal is an endpoint on the Internet or SCN which provides for real-time, two-way communications with another Terminal, or Gateway. This communication may consist of control, indications, audio, and/or data between the two terminals. A terminal may provide indications only, speech only, speech and data, or any combination.

CTT: Communication Terminal Type

CTA: Communication Terminal Address

CTAT: Communication Terminal Address Type

Callable: Capable of being called as described in Section ???. Clients, Servers and Gateways are callable, but Gatekeepers are not.

Endpoint: An H.323 Gateway, CMA client, LDAP server, or CMA Server. An endpoint can call and be called. It generates and/or terminates information streams.

Gatekeeper: The Gatekeeper (GK) is an H.323 entity on the Internet that provides address translation and controls access to the network for H.323 Terminals and Gateways. The Gatekeeper may also provide other services to the H.323 terminals and Gateways, such as bandwidth management and locating Gateways.

Gateway: An H.323 Gateway (GW) is an endpoint on the Internet which provides for real-time, two-way communications between H.323 Terminals on the Network and other ITU Terminals on a wide area network, or to VoIP Clients. Other ITU Terminals include those complying with Recommendations H.310 (H.320 on B-ISDN), H.320 (ISDN), H.321 (ATM), H.322 (GQOS-LAN), H.324 (GSTN), H.324M (Mobile), and V.70 (DSVD).

H.323 Entity: Any H.323 component, including H.323 Terminals, Gateways, Gatekeepers.

Information Stream: A flow of information of a specific media type (e.g. audio) from a single source to one or more destinations.

Internet address: The network layer address of a H.323 or CMA Sys entity as defined by the (inter)network layer protocol in use (e.g. an IP address). This address is mapped onto the layer one address of the respective system by some means defined in the (inter)networking protocol.

Internet: An inter-network of networks interconnected by bridges or routers. LANs described in H.323 may be considered part of such internetworks .

IVR: Interactive Voice Response.

RAS Channel: Unreliable channel used to convey the registration, admissions, bandwidth change, and status messages (following H.225.0) between H.323 entities or CMA Sys entities.

Reliable Channel: A transport connection used for reliable transmission of an information stream from its source to one or more destinations.

Reliable Transmission: Transmission of messages from a sender to a receiver using connection-mode data transmission. The transmission service guarantees sequenced, error-free, flow-controlled transmission of messages to the receiver for the duration of the transport connection.

Soft Link: A referral from one CMA to another.

Subscriber: An "owner" of a CMA. A subscriber can be a person or an organization.

Switched Circuit Network (SCN): A public or private switched telecommunications network such as the GSTN, N-ISDN, or B-ISDN.

Transport Address: The transport layer address of an addressable H.323 entity as defined by the (inter)network protocol suite in use. The Transport Address of an H.323 entity is composed of the LAN address plus the TSAP identifier of the addressable H.323 entity.

Transport Connection: An association established by a transport layer between two H.323 entities for the transfer of data. In the context of H.323, a transport connection provides reliable transmission of information.

TSAP Identifier: The piece of information used to multiplex several transport connections of the same type on a single H.323 entity with all transport connections sharing the same LAN address, (e. g. the port number in a TCP/UDP/IP environment). TSAP identifiers may be (pre)assigned statically by some international authority or may be allocated dynamically during setup of a call. Dynamically assigned TSAP identifiers are of transient nature, i. e. their values are only valid for the duration of a single call.

Unicast: A process of transmitting messages from one source to one destination.

VOIP Environment: The system of Communications Terminals, Servers, Gateways and Gatekeepers for communication over IP networks and interconnection to the SCN.

Well-known TSAP Identifier: A TSAP identifier that has been allocated by an (international) authority that is in charge for the assignment of TSAP identifiers for a particular (inter)networking protocol and the related transport protocols -- (e.g. the IANA for TCP and UDP port numbers). This identifier is guaranteed to be unique in the context of the respective protocol.

1.5 References

The following references contain provisions which are referenced in this text. At the time of publication, the editions indicated were valid.

- [1] ITU-T Recommendation H.225.0 (1996): " Media Stream Packetization and Synchronization for Visual Telephone Systems on Non-Guaranteed Quality of Service LANs ".
- [2] ITU-T Recommendation H.245 (1995): "Control of communications between Visual Telephone Systems and Terminal Equipment".
- [3] ITU-T Recommendation T.120 (1994): "Transmission protocols for multimedia data"
- [4] ITU-T Recommendation H.320 (1995): "Narrow-band ISDN visual telephone systems and terminal equipment".
- [5] ITU-T Recommendation H.321 (1995): "Adaptation of H.320 Visual Telephone Terminals to B-ISDN Environments".
- [6] ITU-T Recommendation H.322 (1995): "Visual Telephone Systems and Terminal Equipment for Local Area Networks which Provide a Guaranteed Quality of Service".
- [7] ITU-T Recommendation H.324 (1995): Terminal for Low Bitrate Multimedia Communications".
- [8] ITU-T Recommendation H.310 (1996): "Broadband audio-visual communications systems and terminal equipment".
- [8] ITU-T Recommendation Q.931 (1993): "Digital Subscriber Signalling System No. 1 (DSS 1) - ISDN User-Network Interface Layer 3 Specification for Basic Call Control".
- [10] ITU-T Recommendation Q.932 (1993): "Digital Subscriber Signalling System No. 1 (DSS 1) - Generic Procedures for the Control of ISDN Supplementary Services".
- [11] ITU-T Recommendation Q.950 (1993): "Digital Subscriber Signalling System No. 1 (DSS 1) - Supplementary Services Protocols, Structure, and General Principles".
- [12] ISO/IEC 10646-1 (1993): "Information Technology - Universal Multiple-Octet Coded Character Set (USC) -- Part I: Architecture and Basic Multilingual Plane".
- [13] ITU-T Recommendation E.164 (1991) "Numbering Plan for the ISDN Era".

?? Add in:

- IETF RFC 822 <email address format>
- IETF <> <LDAP related documents>
- X.500 related documents
- X.509 certificate documents
- SSL/PCT documents
- H.323, RAS related documents
- GUID

2. SYSTEM ARCHITECTURE

2.1 Overview

The CMA system is based on having a CMA for each subscriber to the system. The CMA manages incoming and outgoing calls on behalf of the subscriber. For example, incoming calls can be automatically routed to a given communications terminal or rejected altogether according to caller id, time of day and the location of the callee.

The following diagram illustrates the relationship between the various components of the system: A Call Management Agent Server (CMAS) which manages a group of Call Management Agents (CMA's), and which can be accessed by Call Management Agent Clients (CMAC's) using the Call Management Agent Protocol (CMAP).

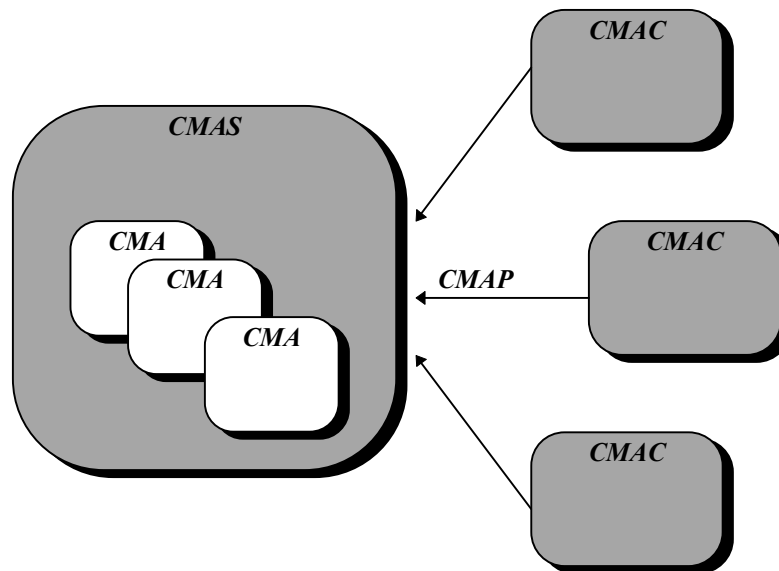


Figure 2-1 CMA System Components

The CMAC's access the CMAS in order to interact with the CMA's managed by the server. Interaction with the CMA's is for the purpose of configuring the CMA or for asking the CMA to perform some service. It is important to note that every CMAS also contains a CMAC component to be able to access other CMAS's.

The following sections define the various components of the system in detail.

2.2 CMA

A CMA is the agent which manages both *outgoing calls* from and *incoming calls* a person's² set of communication terminals. The term "manage" in this context refers to knowing the list of the

² Of course, a CMA can be associated with other entities, such as an *organization*. For sake of simplicity, we shall refer throughout the document to CMA's as belonging to a *person*.

communication terminals through which this person can be accessed, along with relevant attributes such as the communication addresses - And applying some logic to perform decisions as to which devices to route calls to given various input parameters such as the caller id, the time of day, the accessibility of the callee through a given communication terminal, and so forth. Typically, the communication terminal addresses would be IP addresses of Internet Phones, SCN #'s, Pager #'s, etc. A person could have multiple communication terminals of the same type (Such as several phone numbers).

A given CMA is mainly associated with the following elements:

- A CMAS, where it resides.
- At least one *CMAA*, which is used to globally uniquely identify it, and to locate the server where it resides.
- A *list of communication devices* which it manages, along with their various attributes which could be **dynamic**.
- *Call management logic* which is used by the agent to compute to which communication devices to route the caller to given various parameters such as the caller's id, the time of day, the current location of the callee, etc. ***The way the call management logic and its accompanying data is "injected" into a given agent is beyond the scope of this specification.***

To contact someone, you need to contact his CMA (Either directly or through your own CMA), identify yourself (to a controlled level, as you might like to be anonymous), and ask for the (set of) communication terminal address(es) you can use to contact the person. The result might be one or more communication terminal addresses to call, or a *referral to another CMA* you should continue this call setup procedure with.

In some cases, you might ask to communicate with the remote person on a specific type of communication terminal. This could be another (optional) input parameter to the callee's CMA that would be entered when the call is originated.

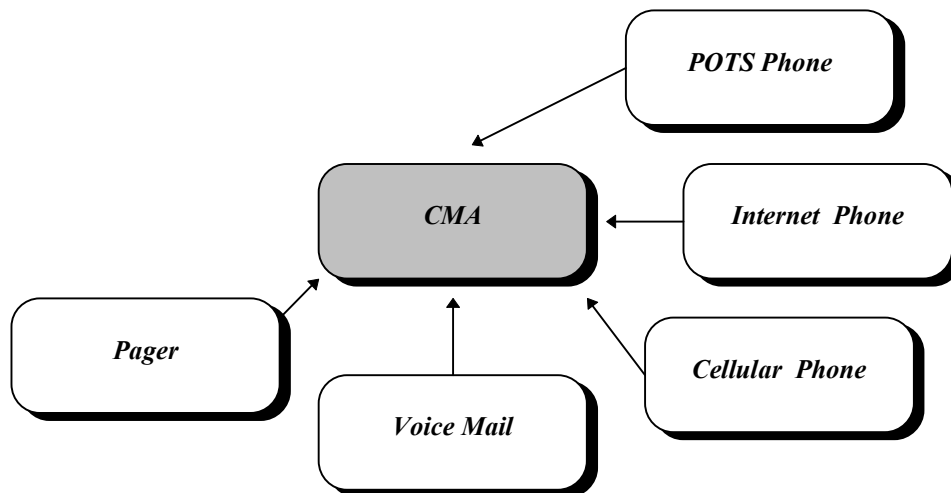


Figure 2-2 The CMA manages multiple communication terminals

The methods and information exposed by the CMA entity are specified in detail in chapter 5.

2.3 CMAA

A CMA is provisioned with *at least* one CMA address - Its CMAA. A CMAA is a globally unique address which is used to locate the CMA throughout the Internet (That is, locate the CMAS where the specific CMA resides at) to be able to interact with it. A subscriber to the CMA system can use his CMAA as a **single, logical communication address**, masking the details of the actual communication terminal used. The need for multiple CMAA addresses arises when confronted with the constraint to provide both Internet friendly and dialpad friendly addresses. Given these parameters, both RFC 822 e-mail and E.164 CMAA types are defined as part of the standard.

Please note the following regarding the E.164 CMAA:

- It could be a real SCN-allocated E.164 (A person's actual primary #)
- It could be a "non existent" E.164 # which has been allocated from a different country code (Such as an Internet country code).
- The E.164 sub-address might need to be used to provide some hints as to the location of the CMAS in charge of the CMA in question (TBD).

2.4 CMAS

The CMA Server (CMAS) is an entity which manages CMA's and is accessible through the CMAP. A given CMA System would typically encompass numerous CMAS's, each managing a given set of CMA's. CMAS's are accessible through the CMAP protocol which originates in CMAC's. As CMA's need to interact with other CMA's (For example, a caller's CMA must interact with the callee's CMA to resolve the needed call setup information), the CMAS includes a CMAC component through which it can interact with other CMAS's over the CMAP protocol.

A CMAS would typically provide a mechanism (which is beyond the scope of this specification) to allow one to "inject" the logic of a given CMA. This logic will control the way the CMA processes incoming and out going call requests.

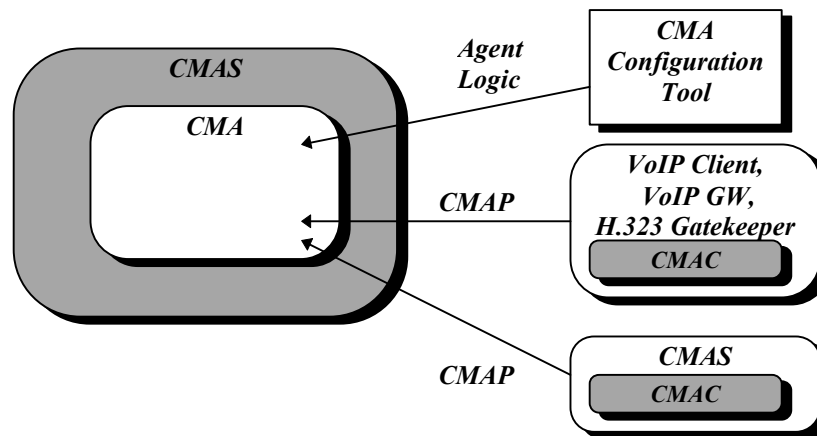


Figure 2-3 Typical CMAS interfaces

2.5 CMAC

A CMA Client (CMAC) is any application which interacts with the CMAS through the CMAP protocol. As illustrated in *Figure 2-3 Typical CMAS interfaces*, the following entities all fall under the CMAC category:

- VoIP client (terminal) applications
- VoIP H.323 SCN Gateways
- VoIP H.323 Gatekeepers
- CMAS servers

It is expected that the CMAC component will be embodied in the form of a library with a well defined API which host application can use to be able to interact with the CMA system. The definition of such an API is beyond the scope of this document although it might be within the scope of separate API definition documents from the VoIP forum (TBD). Such an API is expected to be derived from the method definitions of the CMA which are defined in chapter 5.

2.6 CMAP

The CMA Protocol (CMAP), is the protocol used for client to server and server to server interaction - That is, CMAC to CMAS and CMAS to CMAS³. The protocol syntax and transport are defined in chapter 6.

³ Due to the symmetric system architecture, the latter is actually the same as CMAC to CMAS.

3 . S E C U R I T Y

The following chapter deals with the various security issues relating to the CMA system. Please note that the basic design guideline was to reuse as much as possible from existing standards in this area.

3.1 Authentication

Authentication is a crucial security element of the CMA system, playing a role in:

- CMAS and CMA access control
- CMA call routing computations based on caller id

The authentication element is used to authenticate the identity of a CMAC contacting a CMAS - Either in the case of a client application contacting its CMA or a caller's CMA contacting the callee 's CMA to gather the needed information to setup the call.

It is important to state that the following issues are *not* within the scope of the CMA system authentication element:

- Authenticating the identity of a client application to another client application (This is in the domain of the session control protocol)⁴.

Authentication will be provided by using the SSL secure transport layer (*or some other equivalent such as PCT*). This will also provide a framework for encryption (TBD).

Please note that when a given CMAC contacts a CMAS, it must know beforehand if the transport channel over which it is going to contact the CMAS has to be secure or non secure. For this purpose, different well known ports will be used for secure and non secure access.

3.2 Digital Signatures

Information stored in the CMAS could be optionally digitally signed using X.509 compliant certificates. The exact mechanism is TBD.

3.3 Token Based Access Control

A CMAS could optionally generate an *access token* to be used by the calling client when accessing a service such as a TGW. This token would then have to be passed on through the session control protocol between the two call endpoints.

The exact mechanism for the token generation is TBD. It could be defined in the form of an optional *Token Granting Agent* profile.

⁴ As both the caller's CMA and callee's CMA were involved in the call, it's quite possible to be able to have the CMA system generate certificates or tokens for a given call so both caller and callee can authenticate the identity of the remote party. This issue is extremely important for controlling access to TGWs.

4 . E X T E R N A L I N T E R F A C E S

The following chapter describes the mechanisms through which the CMA system interfaces with other related systems.

4.1 Directory Services: LDAP, X.500

Directory services are needed for the management of static, white-pages like information. The interface between the standard directory services systems such as LDAP and X.500 to the CMA system will be accomplished by extending the standard directory service “user” object schema to including the following additional fields:

- The person’s CMAA.
- An optional pointer to the location of the CMAS in charge of the specific CMA.

Given these additional fields, one will be able to query a directory service and locate the CMAA of the callee, and then go through the chain of CMA related actions to setup the call.

The additional fields syntax is TBD.

?? Hold All CMAA? Only One? Which One?

4.2 H.323 Gatekeeper

The Gatekeeper, which is optional in an H.323 system, provides limited call control services to H.323 Entities, including the following primary services:

- Address Translation
- Admissions Control
- Bandwidth Control

In order to provide CMAC-less H.323 clients (client = terminal or gateway) with the basic needed call information, CMA compliant H.323 Gatekeepers will be needed. A CMA compliant H.323 Gatekeeper will have an H.323 Gatekeeper interface to interact with H.323 clients, and a CMAC to interact with CMAS’s. It will basically translate H.323 RAS protocol requests to/from CMAP requests, allowing seamless integration of pure H.323 clients with the CMA system. The level of functionality which pure H.323 clients will receive from the CMA system is obviously limited to the H.323 Gatekeeper access protocol’s syntax and semantics.

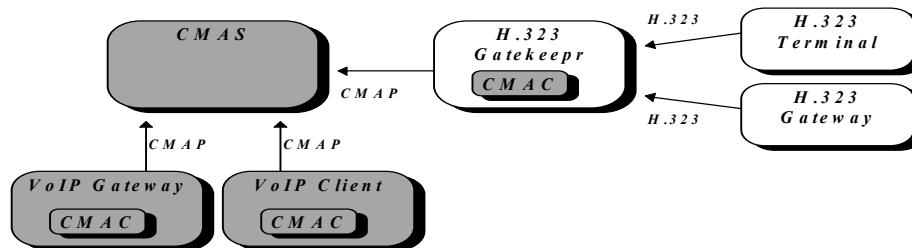


Figure 4-1 H.323 Gatekeeper Interface

The protocol conversion semantics are TBD. The mappings will probably be based on the H.323 RAS **destinationInfo** field.

4.3 VoIP/H.323 Gateway

The H.323 Gateway provides the appropriate translation between transmission formats (for example H.225.0 to/from H.221) and between communications procedures (for example H.245 to/from H.242). The Gateway also performs call setup and clearing on both the Internet side and the SCN side. Translation or conversion between video, audio, and data formats may also be performed in the Gateway. In general, the purpose of the Gateway is to reflect the characteristics of a Internet endpoint to an SCN endpoint, and the reverse, in a transparent fashion.

In the VoIP context, the gateway would focus on translating Internet voice calls to SCN voice calls and vice versa (This would also enable SCN to SCN calls through the Internet). A VoIP gateway will basically be an H.323 based gateway with the addition of a CMA component, enabling it to provide CMA services for calls originating in the SCN domain. Therefore, the gateway extends the types of communication terminals which can use the services of the CMA system to standard SCN terminals such as PSTN telephones.

The SCN user interface provided by the SCN side of the gateway is beyond the scope of this document. Typical user interfaces will be IVR based.

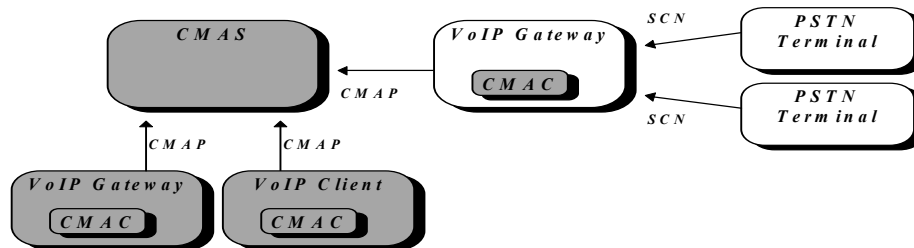


Figure 4-2 H.323/VoIP Gateway Interface

5 . C M A S E M A N T I C S

The following chapter defines the functional and data semantics of the Call Management Agent.

The data maintained by a Call Management Agent consists of the following:

- One or more *CMAAs* (Call Management Agent Addresses) which distinguish this agent from all other agents.
- If this Call Management Agent is simply a redirection pointer or *Soft Link* to another CMA, the CMA contains the CMAA of the target being redirected to. This allows a user to gracefully change his Call Management Agent. For example, this capability would be used to leave a forwarding pointer for a user who changes his E.164 telephone number, or his provider-based E-mail address.
- If this is a terminal Call Management Agent representing a callee, the CMA contains a set of *Communication Terminal Specifiers* (CTSPECS), one for each of the communication devices a VoIP client operating on behalf of this user's CMA has registered with the Call Management Agent Server.

These data types are defined in more detail in the following subsections. CMA Interface methods are defined beginning in section 5.2. The methods are first defined as an informal API-like specification, followed (in section ??) by the formal protocol definitions which constitute the message exchange used to perform the method.

5.1 CMA Object Instance Data

5.1.1 CMAA

A Call Management Agent Address is the fundamental data type which unambiguously identifies this agent. It is a "Name" for the CMA. A Call Management Agent may be identified by more than one address; all of the CMAAs associated with one CMA are considered synonymous. The architecture permits extensible forms of CMAA, however, initially we define two forms which must both be supported by all Call Management Agent Servers. These are:

1. An RFC822 compliant email address, or
2. A fully-qualified E.164 telephone number

Note that either of these is permitted in the **destinationInfo** field of an H.323 gatekeeper interaction.

5.1.2 Target CMA

If this Call Management Agent is a *SoftLink*, the Target CMA contains the CMAA which should be returned to a client as the Call Management Agent to contact instead of this CMA. If a given CMA contains both a Target CMA *and* one or more CTSPECS (see 5.1.3) , the target CMA should be ignored and this CMA be used in answering queries from CMA clients. This behavior allows relatively straightforward failover-based call forwarding in the absence of a more sophisticated agent which performs intelligent call forwarding.

5.1.3 CTSPEC

Communication Terminal Specifications are used by VoIP clients to inform their Call Management agents what their active communication devices are, and what address(es) are currently associated with each terminal. The architecture allows both multiple devices per call management agent and multiple equivalent addresses per terminal. The former capability allows a user to consistently manage multiple devices which may be controlled by different VoIP clients. The latter capability allows for improved availability and performance by permitting a given terminal to be reached multiple ways.

A CTSPEC contains the following information:

5.1.3.1 GUID

The Globally Unique Identifier (GUID) is an identifier which distinguishes this CTSPEC from all others both spatially and temporally. This identifier is generated by the VoIP client which controls or “owns” the communication terminal. Since a GUID generator is expected to be available on most if not all target VoIP platforms, this identifier is a GUID as specified in (*cite: Microsoft GUID Specification here??*).

GUIDs are generated by clients. To prevent aliasing among identical communication devices, a client is expected to remember the GUID for its communication terminal for at least as long as the TTL for the CTSPEC (see section 5.1.3.2), even in the event of a crash.

5.1.3.2 Time-to-Live

Each CTSPEC contains a TTL which is specified by the client and maintained by the Call Management Agent. The client refreshes the TTL by performing a **KeepAlive** method (see section 5.4.2). If the TTL expires, the CTSPEC is silently deleted by the CMA. The TTL is a signed 32 bit integer, in units of seconds. A value of -1 indicates an *infinite* TTL. A CTSPEC with an infinite TTL will never be deleted by the CMA. Infinite TTL’s are useful for communication terminals with static addresses such as POTS telephones and cellular telephones.

5.1.3.3 Terminal Category and Usage Qualifier

The terminal category is a string which describes the general *kind* of communication terminal the CTSPEC refers to. The string is meant to be descriptive and vendor extensible, but with some structure and conventions that enable parsing and interpretation by either VoIP clients or CMA methods that want to do clever things based on the kind of terminal the VoIP client would be communicating with. In order to enforce some basic functionality and consistency in the first deployment of VoIP systems, however, we define a few common terminal categories which all servers and most clients are expected to understand. These are specified below in Table 1:

Table 1: Terminal Categories

Terminal Category	Description
Internet-Phone	A computer-based H.323 audio-only client
Telephone	A normal telephone handset, either POTS, Cellular or ISDN based using E.164 addressing
Fax	A G3 facsimile machine
Pager	A numeric or alphanumeric pager
Voice-Mail	A message-taking service such as a classic PBX-based voicemail system, an H.323 client which records audio messages, etc.
Attendant	A receptionist, secretary, or automated attendant.

These basic categories may be augmented by adding a usage qualifier to the string to further qualify the kind of terminal. As with basic terminal categories, usage qualifiers are vendor extensible, with a few qualifiers specified here for the most common situations. These are defined below in Table 2:

Table 2: Usage Qualifiers

Usage Qualifier	Description
Fixed	Wired, or stays at one location
Mobile	Wireless, or moves frequently from location to location
Business	Used for business communication
Personal	Used for personal communication

This combination of terminal category and terminal qualifier allows devices to be usefully categorized to distinguish, for example, *Personal-Voice-Mail*, from *Business-Voice-Mail*, or a *Business-Mobile-Telephone* from a *Business-Fixed-Telephone*.

5.1.3.4 Terminal Addresses

Associated with a CTSPEC is a set of terminal addresses (CTAddress) that can be used to communicate with that terminal. The addresses contain all of the information needed by the session protocol to initiate a session with the terminal, either directly as in the case of an Internet Phone application or H.323 conferencing application, or indirectly through a VoIP gateway to a PSTN or other telephony service. A CTAddress consists of a pair {address-type, value} as defined below. Where a CTSPEC contains more than one CTAddress, VoIP clients wishing to communicate with that terminal will interpret the set of addresses as being:

- a) *equivalent* - all the addresses in fact reach the same terminal, and
- b) *ordered* - the addresses are in decreasing order of preference from the point of view of the call management agent returning the information.

This allows a VoIP client, in cooperation with a Call Management Agent Server, to provide capabilities such as priority ordering of terminal addresses or load balancing among gateways (by explicitly randomizing the list order each time it is returned to a client).

Address types specify the syntax and usage for the CTAddress. An Address Type is a string, to allow for easy extensibility to new addressing forms. In order to enforce some basic functionality and consistency in the first deployment of VoIP systems, however, we define a few common address types which all servers and most clients are expected to understand. These are specified below in Table 3.

Table 3: CTSPEC Address types

Address Type	Value Syntax	Description
H.323	H.323 transportAddress	The destCallSignalAddress of an H.323 compliant VoIP client terminal (or VoIP gateway to the terminal)
E.164	digit string	A fully qualified E.163 telephone number
DNS	ASCII string	an IP host name or RFC822 email address. This can also be used as the destinationInfo for initiating an H.323 session.

?? In the DNS case, the address value needs to optionally convey the PORT and transport type - TCP, UDP, Secure TCP, etc.

5.1.3.5 Capability List

Associated with a CTSPEC is a capability list for the corresponding terminal. The capability list is intended to give the general capabilities of the terminal, so a potential caller can decide whether or not it is worth trying to communicate with the end user through this terminal. Therefore, it is used to indicate such things as “video capable”, or “data conferencing capable” rather than to replace low level or detailed capabilities exchange such as is done via H.245 at session establishment time. A capability list is a string, to allow for easy extensibility to new addressing forms. The string syntax is a comma-separated list of tokens, where each token can be either:

- a keyword, such as “video”, or
- a keyword=value, such as *<anybody got a good example...>*

In order to enforce some basic functionality and consistency in the first deployment of VoIP systems, however, we define a few common keywords which all servers and most clients are expected to understand. These are specified below in Table 4.

Table 4: Capability list keywords

Keyword	Description
Video	The terminal can do real time video
Data	The terminal can do T.120 compliant data conferencing
ReceiveOnly	The terminal is only capable of receiving (e.g. a pager, radio, television, etc.)

5.2 Call Management Agent Method Summary

Table 5 below summarizes the methods which operate on a CMA and cross-references the section in which the method is defined.

Table 5: CMA Method Summary

Method	Description	Reference
CreateCMA	Creates a new Call Management Agent	5.3.1
RemoveCMA	Remove an existing Call Management Agent	5.3.2
AddSynonym	Adds a synonymous address to a Call Management Agent	5.3.3
RemoveSynonym	Removes an address from a Call Management Agent and deletes the CMA if this is the last address associated with the CMA.	5.3.4
SetTargetCMAA	Associates a <i>SoftLink</i> target Call Management Agent with a CMA.	5.3.5
GetTargetCMAA	Returns the <i>SoftLink</i> target Call Management Agent Address associated with a CMA, if one exists	5.3.6
UnSetTargetCMAA	Disassociates a <i>SoftLink</i> target Call Management Agent Address, if one exists, from the CMA	5.3.7
SetCTSpecifier	Enters one or more new Communication Terminal Specifiers for a Call Management Agent, or replaces the data for an existing CTSPEC.	5.4.1
KeepAlive	Performs a refresh of the Time-To-Live of an existing Communication Terminal Specifier.	5.4.2
Resolve	Obtains the set of usable Communication Terminal Specifiers that the calling client can use to contact the VoIP client(s) associated with a given CMA.	5.5.1
GetSynonyms	Returns all of the Call Management Agent Addresses currently associated with the specified CMA.	5.5.2

5.3 Methods for Creating and Maintaining CMAs

The following methods are used to create a CMA, remove a CMA, associate or disassociate a synonymous CMAA with a CMA, and create or remove a CMA *Softlink*.

5.3.1 CreateCMA

A new CMA is created using this method and providing a CMAA which identifies this particular call management agent. The CMA continues to exist until the last CMAA is disassociated with it (see section 5.3.2 for more information).

The arguments to **CreateCMA** are as follows:

Argument	Value Syntax	In/Out	Description
agentCMAA	String	In	The address that the new agent is to be known by.

The **CreateCMA** method can generate the following return values:

Return Code	Description
Success	the requested agent has been created
NoCommunication	Some communication-oriented failure occurred, e.g. a required CMAS was down.
existingCMAA	An agent with this CMAA already exists
AuthenticationFailure	One or more CMA servers do not believe the caller's identity

Return Code	Description
AccessDenied	The caller does not have permission to create the requested CMA.

5.3.2 RemoveCMA

A CMA is removed using this method and providing a CMAA which identifies this particular call management agent.

The arguments to **RemoveCMA** are as follows:

Argument	Value Syntax	In/Out	Description
agentCMAA	String	In	The address of the CMA to remove.

The **RemoveCMA** method can generate the following return values:

Return Code	Description
Success	the requested agent has been removed
NoCommunication	Some communication-oriented failure occurred, e.g. a required CMAS was down.
NonExistingCMAA	An agent with this CMAA doesn't exist
AuthenticationFailure	One or more CMA servers do not believe the caller's identity
AccessDenied	The caller does not have permission to remove the requested CMA.

5.3.3 AddSynonym

This method adds a CMAA to an existing CMA. The method is *idempotent* — it returns success whether or not the synonymous CMAA is already one of the CMAAs associated with this CMA.

The arguments to **AddSynonym** are as follows:

Argument	Value Syntax	In/Out	Description
agentCMAA	String	In	One of the existing CMAAs for the agent the synonym is to be associated with
newCMAA	String	In	The synonymous CMAA to be associated with the agent

The **AddSynonym** method can generate the following return values:

Return Code	Description
Success	the requested synonym has been added
NoCommunication	Some communication-oriented failure occurred, e.g. a required CMAS was down.
nosuchCMAA	An agent with this CMAA does not exist.
conflictingCMAA	An different agent with this CMAA already exists
AuthenticationFailure	One or more CMA servers do not believe the caller's identity
AccessDenied	The caller does not have permission to modify the CMA in this way

5.3.4 RemoveSynonym

This method removes a CMAA from an existing CMA. If this is the only CMAA currently associated with a CMA, the CMA object is deleted as a side-effect.

The arguments to **RemoveSynonym** are as follows:

Argument	Value Syntax	In/Out	Description
----------	--------------	--------	-------------

agentCMAA	String	In	The CMAA to be removed from the set of CMAAs that the agent is known by.
deleted	BOOLEAN	Out	set TRUE if the RemoveSynonym method removed the only remaining CMAA of a CMA.

The **RemoveSynonym** method can generate the following return values:

Return Code	Description
Success	the requested synonym has been removed
NoCommunication	Some communication-oriented failure occurred, e.g. a required CMAS was down.
nosuchCMAA	An agent with this CMAA does not exist.
AuthenticationFailure	One or more CMA servers do not believe the caller's identity
AccessDenied	The caller does not have permission to modify the CMA in this way

5.3.5 SetTargetCMAA

This method associates a *SoftLink* target CMAA with a CMA. If a target CMAA is present in a call management agent and the agent has no current CTSPECS associated with it, a query on this CMA will return the target CMAA in a *redirect* to the requesting client. Note that the target CMAA does not necessarily have to exist either at the time this call is made, nor later, since multiple servers may be involved and dangling pointers are always a hazard in such systems.

The arguments to **SetTargetCMAA** are as follows:

Argument	Value Syntax	In/Out	Description
agentCMAA	String	In	One of the existing CMAAs for the agent the target is to be associated with
targetCMAA	String	In	The target CMAA this CMA is to be <i>SoftLinked</i> to.

The **SetTargetCMAA** method can generate the following return values:

Return Code	Description
Success	The requested target has been set.
NoCommunication	Some communication-oriented failure occurred, e.g. a required CMAS was down.
nosuchCMAA	An agent with this CMAA does not exist.
AuthenticationFailure	One or more CMA servers do not believe the caller's identity
AccessDenied	The caller does not have permission to modify the CMA in this way

5.3.6 GetTargetCMAA

This returns the *SoftLink* target Call Management Agent Address associated with a CMA, if one exists.

The arguments to **GetTargetCMAA** are as follows:

Argument	Value Syntax	In/Out	Description
agentCMAA	String	In	One of the existing CMAAs for the agent the target is to be associated with
targetCMAA	String	Out	The target CMAA this CMA is <i>SoftLinked</i> to.

The **GetTargetCMAA** method can generate the following return values:

Return Code	Description
Success	The requested target has been set.

Return Code	Description
NoCommunication	Some communication-oriented failure occurred, e.g. a required CMAS was down.
nosuchCMAA	An agent with this CMAA does not exist.
noTargetCMAA	This CMA does not have a target associated with it
AuthenticationFailure	One or more CMA servers do not believe the caller's identity
AccessDenied	The caller does not have permission to examine the CMA in this way

5.3.7 UnsetTargetCMAA

This method disassociates a *SoftLink* target CMAA, if one exists, from the CMA.

The arguments to **UnSetTargetCMAA** are as follows:

Argument	Value Syntax	In/Out	Description
agentCMAA	String	In	One of the existing CMAAs for the agent the target is to be associated with

The **UnSetTargetCMAA** method can generate the following return values:

Return Code	Description
Success	The requested target has been set.
NoCommunication	Some communication-oriented failure occurred, e.g. a required CMAS was down.
nosuchCMAA	An agent with this CMAA does not exist.
AuthenticationFailure	One or more CMA servers do not believe the caller's identity
AccessDenied	The caller does not have permission to modify the CMA in this way

5.4 Methods for maintaining CTSPECS

The following methods are used to maintain CTSPECS. We assume that each VoIP client has access to a local *GUID* generator that can be used to get that GUID for a new CTSPEC, and that the client has sufficient non-volatile storage to remember a GUID for at least the time period he specifies as the **TTL** of the CTSPEC.

5.4.1 SetCTSpecifier

This method enters one or more new CTSPECS for a CMA, or replaces the data for an existing CTSPEC. The CTSPEC to be created or modified is identified by the **GUID** field in each **CTSPEC** argument.

A CTSPEC can be deleted using this call by specifying a **TTL** of zero for the CTSPEC.

If the call fails, *none* of the requested CTSPECS have been either added, modified, or removed.

The arguments to **SetCTSpecifier** are as follows:

Argument	Value Syntax	In/Out	Description
agentCMAA	String	In	One of the existing CMAAs for the agent the target is to be associated with
ctSpecs	array of ctSpec	In	The set of CD specifiers which can be used to communicate with the callee identified by the agentCMAA .

The **SetCTSpecifier** method can generate the following return values:

Return Code	Description
Success	the ctSpecs have been added and/or modified
NoCommunication	Some communication-oriented failure occurred, e.g. a required CMAS

Return Code	Description
NoSuchCMAA	was down.
AuthenticationFailure	The agentCMAA does not exist or cannot be located.
AccessDenied	One or more CMA servers do not believe the caller's identity The caller does not have permission to modify the CMA in this way

5.4.2 KeepAlive

This method performs a refresh of the TTL of an existing CTSPEC. It performs two related functions for VoIP clients:

1. It provides a low-overhead method of refreshing the **TTL** of a CTSPEC, since it is intended to be more efficient than periodically doing a **SetCTSspecifier**.
2. It provides a VoIP client with some assurance that it has sufficient network connectivity to be able to receive calls.

The arguments to **KeepAlive** are as follows:

Argument	Value Syntax	In/Out	Description
agentCMAA	String	In	One of the existing CMAAs for the agent the target is to be associated with
guid	GUID	In	The GUID of the CTSPEC to be refreshed.

The **KeepAlive** method can generate the following return values:

Return Code	Description
Success	the ctSpec corresponding to this guid has had its TTL refreshed.
NoCommunication	Some communication-oriented failure occurred, e.g. a required CMAS was down.
NoSuchCMAA	The agentCMAA does not exist or cannot be located.
NoSuchCTSPEC	No CTSPEC associated with this CMA has a matching guid .
AuthenticationFailure	One or more CMA servers do not believe the caller's identity
AccessDenied	The caller does not have permission to modify the CMA in this way

5.5 Methods for Obtaining information from a CMA

VoIP clients wishing to make calls use the following methods to obtain information about other VoIP endpoints, and especially to obtain the (often dynamic) addressing information needed by the session protocol to establish communication with another VoIP endpoint.

5.5.1 Resolve

This method obtains the set of usable CTSPECS that the calling client can use to contact the VoIP client(s) associated with a given CMA. It takes as input two CMAAs: the CMAA of the requesting client, and the CMAA of the callee the client wishes to contact. The CMAA of the requesting client is used when the requester wishes his own call management agent to be involved in computing the result of the query. The method returns the subset of the callee CMA's CTSPECS deemed usable by both the callee's and the caller's CMAs. As noted earlier, this list is to be treated as ordered in decreasing preference.

A CTSPEC returned by **Resolve** may be cached by a client no longer than the **TTL** in the CTSPEC.

The arguments to **Resolve** are as follows:

Argument	Value Syntax	In/Out	Description
callerCMAA	String	In	The address of the requester's Call Management

calleeCMAA	String	In/Out	Agent. If not present or null, the caller's CMA is not involved in computing the results of the Resolve operation. The address of the requested callee's call management agent. Note that the value returned may be different from the value supplied if a <i>Softlink</i> is traversed.
anonymous	Boolean	In	Do not reveal the caller's identity to any CMA other than the CMA identified by the callerCMAA argument.
ctSpecs	array of ctSpec	Out	The set of CT specifiers which can be used to communicate with the callee identified by the calleeCMAA .

The **Resolve** method can generate the following return values:

Return Code	Description
Success	A (possibly null) ctSpecs has been returned
NoCommunication	Some communication-oriented failure occurred, e.g. a required CMAS was down.
NoSuchCallerCMAA	The callerCMAA does not exist or cannot be located.
NoSuchCalleeCMAA	The calleeCMAA does not exist or cannot be located.
AuthenticationFailure	One or more CMA servers do not believe the caller's identity

5.5.2 GetSynonyms

This method returns all of the CMAAs currently associated with the specified CMA.

The arguments to **GetSynonyms** are as follows:

Argument	Value Syntax	In/Out	Description
agentCMAA	String	In	One of the existing CMAAs for the agent the synonym is to be associated with
synonymCMAAs	array of Strings	Out	The complete set of synonymous CMAAs currently associated with the agent

The **GetSynonyms** method can generate the following return values:

Return Code	Description
Success	The requested synonym has been added
NoCommunication	Some communication-oriented failure occurred, e.g. a required CMAS was down.
nosuchCMAA	An agent with this CMAA does not exist.
AuthenticationFailure	One or more CMA servers do not believe the caller's identity

6. PROTOCOL SYNTAX

6.1 Introduction

The following chapter provides the ASN.1 syntax specification of the CMA Protocol used for client-server and server-server communication.

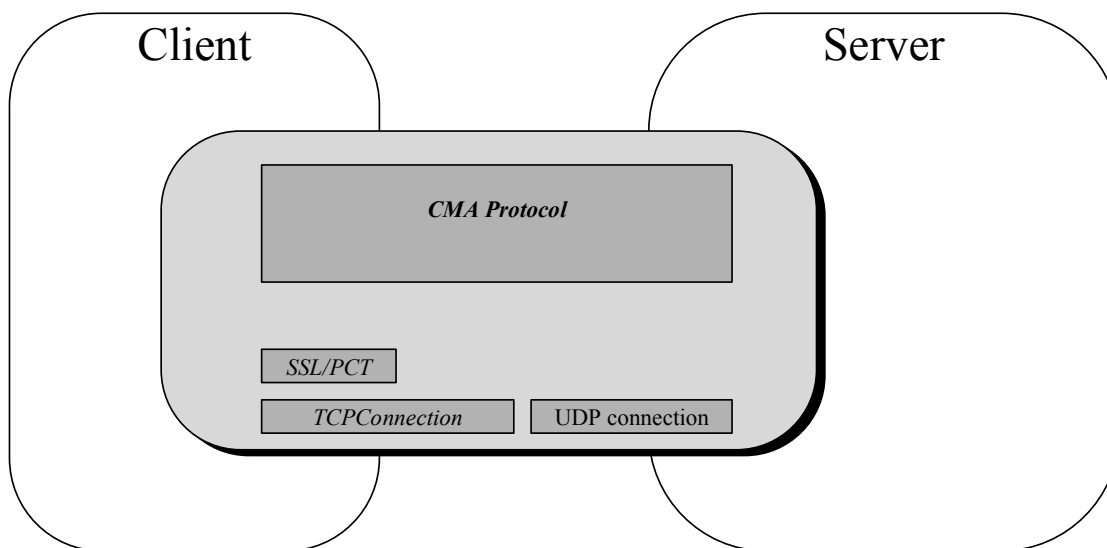


Figure 6-1 The CMAP stack layout

The *CMAP Protocol* is a simplified protocol for agent activation. This protocol can be considered the “carrier” protocol in the sense that it provides an extensible platform for accessing additional types of agents beyond the pure Call Management Agent.

6.2 Protocol Transport

Currently, the protocol is defined to work on TCP/IP connections to the well know TCP port (*TBD*).

A UDP based protocol might also be used. The UDP packet would contain the list of commands to execute. Logically, the connection is opened and closed in a single request. In general, UDP should be limited to short query operations because of its unreliable nature.

A PCT/SSL-based transport will be used for authenticated and secured connections. Secure connections will work over the well known TCP port (*TBD*).

After a connection is established, and the connect command is recognized, the application layer will check the authentication and encryption used to open the channel. If they are insufficient, the request will be rejected. In this case, the client will either fail, or re-open the connection with a higher level of authentication.

6.3 General Protocol Notes

Generally, the PDUs (Program Data Units) used in the protocol suite are divided into two types - requests and replies. All requests have a request ID and all replies carry within them the request ID so the reply could be associated with the request it answers.

All packets arriving at an CMAP server should be of a type called CMAPCOMMANDS. The packet is described in the ASN.1 notation below:

```
CMAPCOMMANDS ::= CHOICE
{
  -- command PDUs:
  getprop-pdu          GETPROP-PDU,
  setprop-pdu          SETPROP-PDU,
  activatemethod-pdu  ACTIVATEMETHOD-PDU,
  createinstance-pdu  CREATEINSTANCE-PDU,
  deleteinstance-pdu  DELETEINSTANCE-PDU,

  -- reply PDUs:
  redirect-pdu         REDIRECT-PDU,
  getprop-reply-pdu   GETPROP-REPLY-PDU,
  setprop-reply-pdu   SETPROP-REPLY-PDU,
  activatemethod-reply-pdu ACTIVATEMETHOD-REPLY-PDU,
  createinstance-reply-pdu CREATEINSTANCE-REPLY-PDU,
  deleteinstance-reply-pdu DELETEINSTANCE-REPLY-PDU,
}
```

The following chapters will describe the protocols by specifying in detail each of the ASN.1 PDUs used in each protocol.

6.4 Connect sequence

6.4.1 Overview

Before any CMAP operation could be performed, a CONNECT-PDU must be sent by the client. It contains the protocol identifier used by the client. The server decides based on that information whether or not to accept the attempted connection, and which level of protocol to expose to the client.

The client does not wait for a reply for the Connect command - it immediately sends the first command to the server.

There is no "ACK" for the connect request. If the server replies to the command following the Connect, then the connect was accepted. If the first reply was CONNECT-NACK, then it means no further command following the Connect will get a reply.

6.4.2 PDU structure

6.4.2.1 CONNECT-PDU

```
CONNECT-PDU ::= SEQUENCE {
  protoIdent OBJECT IDENTIFIER,
  application OCTET STRING OPTIONAL,
  nonStandard SEQUENCE {
    id OBJECT IDENTIFIER,
    ...
  } OPTIONAL
```

}

As explained above, the connect pdu should be the first pdu to arrive from a client on every attempted CMAP session (otherwise the server will automatically reject the calling client). The connect PDU contains the protocol version used, and optionally client **application** information.

A non-standard structure allows for vendor-specific feature announcement.

The **protoIdent** protocol identifier should be **??voip.cma.protocol.major(1).minor(0)??**.

The protocol minor version will be incremented with each update of the protocol, while a compatibility is maintained with previous version. When the new version is incompatible with the previous one, the major version will be incremented. Thus, the general rule is that a server will accept a command with the same protocol major version, regardless of the minor version. It still might use the minor version code to “optimize” its replies to this level. If the server supports a higher major version, but still support previous one, (for backward compatibility), it will reply with a major version the same as the client’s, possibly with a higher minor version.

Note that there is no **connect - ack** PDU. If the server answers a request, the the connect was successful.

If the connect protocol decides to reject the attempted connection (usually because of incompatible major versions) then a CONNECT-NAK-PDU is sent.

6.4.2.2 CONNECT-NAK-PDU

```
CONNECT-NAK-PDU ::= SEQUENCE {
    reason ENUM ( version-mismatch, busy, access-denied )
}
```

The CONNECT-NAK pdu is sent when the connect protocol decides to reject an attempted connection.

The reason of the rejection is sent in the **reason** field.

6.5 Basic CMAP operations

6.5.1 Overview

Generally, the CMAP protocol provides the application layer with the ability to perform the following operations on an agent residing on the server's agent directory:

- Create a new instance (of a given agent class)
- Delete an instance
- Get data properties of a given instance
- Set data properties of a given instance
- Activate a method of a given instance (with a set of given parameters)

Each of these operations is basically composed of a *request* sent to the server side, and a corresponding asynchronous *reply* sent from the server back to the client.

An additional important reply is the *redirect* reply, which is used by the server to refer requesting clients to other servers, or to another destination object.

All request packets have the following common fields:

- Each pdu starts with a **Protocol identifier**. This identifier should be the same as used by the Connect command (although it can use a sub-protocol of that major-version)
- The **Agent Instance Id** which identifies the agent instance on which the operation should be performed.
- The client **Request Id** which is used by the client to uniquely identify the outgoing request so it could handle multiple asynchronous requests on a single connection.
- An optional **Session Id** which is used for synchronization purposes. It is created and supplied by the server side, through application specific means (By using the *activate method* command). The need for such an id arises in situations where multiple copies of the **same** agent instance can exist in different, replicated servers. This allows the client and server to make sure they are speaking about the same physical agent instance. The client should save the last session-id returned by the server, and return it to the server on future invocations.
- A **destination**, which is the agent on which to operate, or to active the method of. The value of this parameter is carried on from previous command (on the same connection), so it need to be specified only once in a series of commands on the same object.

Please note that the basic CMAP protocol is generic and doesn't discuss any *agent class specific* definitions. Profiles of specific agent class definitions should define:

- Which methods are used (and their corresponding parameters).
- Which result codes are used.
- Which attributes are used.

6.5.2 PDU Structure

6.5.2.1 CMAP Agent Identifier

A CMAP agent identifier is built out of 3 fields:

- **class** - the actual class of this agent, whether it is a call management agent, a token generation agent, etc. The actual classes are defined by a given, specific agent profile. In this context, we'll be defining the call management agent profile in chapter ??.
- **namespace** - an identifier for the namespace used by the actual instance address.
- **instance** - the actual instance address, within the current namespace.

Both class, subclass and namespace are OBJECT IDENTIFIERS, defined under the VoIP OID hierarchy. The instance address is a string (e.g: E.164 phone number, H.323-id).

6.5.2.2 General ASN.1 definitions

6.5.2.2.1 General

```
-- a basic definition of a string in the CMAP
CMAPString ::= IA5String,

-- CMA address: either H.323-id or E.164 number.
CMAA ::= CMAPString,
```

6.5.2.2.2 Agent identifier

An Agent Identifier is used to identify the agent that will be used to carry on the given command. The agent is identified by its **class** and **instance**. The instance id is the unique instance identifier of this agent within this class. The basic agent class defined in the context of this document is of class **call manager agent**. The instance id of the call management agent is its CMAA.

An optional **subclass** vendor-specific identifier may be used to request specific behaviour from this class.

```
rfc822Namespace OBJECT IDENTIFIER ::= { ?? voip . cma . namespace . 1 },
e164Namespace OBJECT IDENTIFIER ::= { ?? voip . cma . namespace . 2 },

classCallMgmtAgent OBJECT IDENTIFIER ::= { ?? voip . cma . class . callmgmt(1)
},

-- CMAP agent identifier
CMAPAgent ::= {
    class          OBJECT IDENTIFIER,
    subclass       OBJECT IDENTIFIER OPTIONAL,
    namespace      OBJECT IDENTIFIER,
    instance       CMAPString,
}
```

6.5.2.2.3 Properties

Each command carries a list of properties. Each property has its id (**PID**) and its **Value**.

PID is a 32-bit integer code.

A "request" command carries a **PIDList**, which is an array of such ids.

A “reply” command carries a **PackedPropertyList**, which holds the Ids with their values.

The following ASN.1 definitions define the various support Property value types:

```
-- property identifier: 32bit integer
PID ::= INTEGER,

-- supported value types for properties:
PackedValue ::=
CHOICE {
    -- value type is unknown
    packedUNKNOWN      [0]  IMPLICIT INTEGER,

    -- this indicates that the type is valid, but the value isn't
    packedInvalid      [1]  IMPLICIT NULL,

    packedLPSTR        [2]  IMPLICIT OCTET STRING,
    packedBYTE         [3]  IMPLICIT INTEGER,
    packedWORD         [4]  IMPLICIT INTEGER,
    packedDWORD        [5]  IMPLICIT INTEGER,
    packedBSTR         [6]  IMPLICIT OCTET STRING,
    packedBOOL         [7]  IMPLICIT INTEGER,

    -- for a nested PROPLIST inside a PROPLIST
    packedPROPLIST     [12] PackedPROPLIST
}

PackedProperty ::= SEQUENCE {
    pid  PID,
    value PackedValue
}

PIDList ::= SEQUENCE OF PID,
PackedPropertyList ::= SEQUENCE OF PackedProperty,

-- various error status codes
PropertyStatus ::= ENUM ( ok(0), not-found, no-access, unkown-error ),
```

6.5.2.3 GETPROP-PDU

```
GETPROP-PDU ::= SEQUENCE
{
    protoIdent  OBJECT IDENTIFIER,
    dest        CMAPAgent,
    req-id      INTEGER,
    pidlist     PIDList,
    session-id  SessionID OPTIONAL
}
```

The getprop pdu is used to get a list of attributes associated at a specific time with a specific instance of a CMA agent.

The information describing the requested CMA agent instance is passed as the *dest* (destination) field. The request is also assigned a request number which is passed as the *req-id* (as described in the overview section above). The **pidlist** field contains a list of the ids of the agent’s properties which we want to know.

The Server replies to the getprop query with a getprop-reply pdu as described below:

6.5.2.4 GETPROP-REPLY-PDU

```
GETPROP-REPLY-PDU ::= SEQUENCE
{
    protoIdent  OBJECT IDENTIFIER,
    dest        CMAPAgent OPTIONAL,
    reply-id    INTEGER,
    status      PropertyStatus,
    props       PackedPropertyList,
    session-id  SessionID OPTIONAL
}
```

As before the getprop-reply-pdu contains the **dest** fields which contains the information about the CMA agent instance whose properties are returned.

The **reply-id** is exactly the same as the reqst-id (req-id field) of the getprop-pdu which we are answering. The **status** field indicates whether or not the query operation was successful. The status field indicates either success, failure or partial-success (not all requested properties could be retrieved).

The **props** field contain a list of all the requested properties (with their values) which could be retrieved from the server.

As explained in the overview, if this operation is related to a specific session (i.e. to a specific "login" operation) it may contain a **session-id** field, indicating to which session this pdu is related.

6.5.2.5 SETPROP-PDU

```
SETPROP-PDU ::= SEQUENCE
{
    protoIdent  OBJECT IDENTIFIER,
    dest        CMAPAgent OPTIONAL,
    req-id      INTEGER,
    props       PackedPropertyList,
    session-id  SessionID OPTIONAL
}
```

The setprop-pdu is used to set a set of properties of an CMAP agent instance to a set of given values. The pdu is very similar to the getprop-pdu and actually differs only in one field. The getprop pdu contains an PIDList which is a list of properties to be retrieved from the server while the setprop pdu contains PackedPropertyList field which contains a property list which will be associated with an agent instance and later retrieved using the getprop-pdu.

The SETPROP-PDU is answered by a SETPRPOP-REPLY-PRU described below.

6.5.2.6 SETPROP-REPLY-PDU

```
SETPROP-REPLY-PDU ::= SEQUENCE
{
    protoIdent  OBJECT IDENTIFIER,
    dest        CMAA OPTIONAL,
    reply-id    INTEGER,
    status      PropertyStatus,
    failed-props  PIDList,
    session-id  SessionID OPTIONAL
}
```

This pdu is a reply to the setprop-pdu described above. This PDU contains the usual **dest**, **reply-id** and **session-id** fields.

The setprop reply also contains a **status** field. This field contains the status of the set operation. If not all properties could be set, then a non-zero status is returned.

When an error status is returned, the field a **failed-props** (which may be empty) is also returned - containing the property IDs of the properties that could not be set.

6.5.2.7 ACTIVATEMETHOD-PDU

```
ACTIVATEMETHOD-PDU ::= SEQUENCE
{
    dest          CMAPAgent OPTIONAL,
    req-id        INTEGER,
    method        CMAPString,
    params        PackedPropertyList,
    session-id    SessionID OPTIONAL
}
```

The ACTIVATEMETHOD-PDU is used to invoke a specific method on a given agent instance on the server. The PDU contains the usual **dest**, **req-id** and **session-id** fields (as described in the overview and in the getprop-pdu).

The activatemethod-pdu also contains a **method** field which is the name of the specific agent method⁵ to invoke on the given agent instance. This method can be either a *class* method or an *instance* method. There's no distinction between these two method categories on the protocol level.

The pdu also contains a **params** field which contains a list of parameters to be passed to the method to be activated.

The activatemethod-pdu is answered by an ACTIVATEMETHOD-REPLY-PDU as described below.

6.5.2.8 ACTIVATEMETHOD-REPLY-PDU

```
ACTIVATEMETHOD-REPLY-PDU ::= SEQUENCE
{
    protoIdent    OBJECT IDENTIFIER,
    dest          CMAPAgent OPTIONAL,
    reply-id      INTEGER,
    status        PropertyStatus,
    params        PackedPropertyList,
    session-id    SessionID OPTIONAL
}
```

The ACTIVATEMETHOD-REPLY-PDU contains the usual **dest**, **reply-id** and **session-id** fields.

The **status** field indicates whether or not the method activation on the server was successful or not.

Note that the status is not the "return value" from the method.

Any value the method returns is passed back as a parameter in the **params** list. The list can also be empty, if no return value is required.

6.5.2.9 CREATEINSTANCE-PDU

```
CREATEINSTANCE-PDU ::= SEQUENCE
{
    protoIdent    OBJECT IDENTIFIER,
    dest          CMAPAgent OPTIONAL,
    req-id        INTEGER,
    session-id    SessionID OPTIONAL
}
```

```
}

```

The CREATEINSTANCE-PDU is used to create an instance of a specific CMA agent on a specific server. The only fields in the createinstance pdu are the **dest** field which indicates the agent which we want to create an instance of and the server we want to create it on. The **req-id** and **session-id** are the same as in the other CMAP PDUs.

This pdu is answered by the CREATEINSTANCE-REPLY-PDU described below

6.5.2.10 CREATEINSTANCE-REPLY-PDU

```
CREATEINSTANCE-REPLY-PDU ::= SEQUENCE
{
    protoIdent  OBJECT IDENTIFIER,
    dest        CMAPAgent OPTIONAL,
    reply-id    INTEGER,
    status      PropertyStatus DEFAULT ( ok ),
    session-id  SessionID OPTIONAL
}

```

The CREATEINSTANCE-REPLY-PDU contains the usual dest, reply-id and session-id fields, and in addition a status field containing a status code for the attempted agent creation.

6.5.2.11 DELETEINSTANCE-PDU

```
DELETEINSTANCE-PDU ::= SEQUENCE
{
    dest        CMAPAgent OPTIONAL,
    req-id      INTEGER,
    session-id  SessionID OPTIONAL
}

```

The DELETEINSTANCE-PDU is used to delete a specific instance of a CMA agent from a specific CMAP server. The only parameters needed by the server for this operation are the **dest**, **req-id** and **session-id**.

This pdu is answered by the DELETEINSTANCE-REPLY-PDU described below.

6.5.2.12 DELETEINSTANCE-REPLY-PDU

```
DELETEINSTANCE-REPLY-PDU ::= SEQUENCE
{
    dest        CMAPAgent OPTIONAL,
    reply-id    INTEGER,
    status      PropertyStatus DEFAULT ( ok ),
    session-id  SessionID OPTIONAL
}

```

Just like in the CREATEINSTANCE-REPLY-PDU this pdu contains the usual **dest**, **reply-id** and **session-id** fields, and a status field indicating the status of the attempted deleteinstance operation.

6.5.2.13 REDIRECT-PDU

```
REDIRECT-PDU ::= SEQUENCE
{
    protoIdent  OBJECT IDENTIFIER,
    reply-id    INTEGER,
    status      ENUM ( permanent, temporary),

    --either new-server or new-dest (or both) MUST appear.
    new-server  OCTET STRING OPTIONAL,
    new-dest    CMAPAgent OPTIONAL,
}

```

}

The REDIRECT-PDU is a reply which could be initiated by any CMAP server as either a single or an extra reply to any pdu it receives. The redirect pdu is intended to cause the client contacting a specific server to redirect its operation to a different CMAP server.

This pdu contains a **reply-id** to indicate which pdu caused this redirect pdu to be sent. The **status** indicates which kind of a redirect this is (e.g. **permanent** - redirect all similar requests to the new server, **temporary** - just for the context of the given request). The **new-server** field contains the address of the new server to contact - either the DNS name or the dotted-decimal notation of its address, followed by colon and the port to use (e.g: "127.0.0.1:123" or "cma.domain.com:3221").

The **new-dest** holds a new CMAP agent instance, to be used.

6.6 CMA Specific Profile

The following section defines the specific call management agent protocol profile in terms of the basic CMAP operations. This section is a projection of the method definitions and data attributes which are defined in a rather free form manner in chapter 5.

TBD - This is a rather mechanical process, we'll perform when the semantics are stabilized.