VOCAL

Vovida Open Communication Application Library

VOCAL Prepaid B2BUA

Software Version 0.1.0 (Alpha)

Revision History The following table itemizes the revision history of this manual:

Table 1. Software Release History

Software Version	***Guide Version***	***Date***	***Comments***
0.1.0	D	9/26/01	Alpha release.

Table 2. Documentation History

Revision	***Date***	***Originator***	***Comments***
A	6/15/01	Chok Lam	Created this document and updated Surendra's initial draft.
B	8/3/01	Chok Lam	Incorporated Surendra's call flow diagrams, state machines and other material.
C	9/23/01	Surendra Prajapat, Chok Lam	Update for Alpha Release
D	9/26/01	David Kelly	Original word document ported into FrameMaker and edited by David Kelly
E	10/8/01	David Kelly	Added descriptions for MLCC state machine and RADIUS messages.
F	10/9/01	David Kelly	Added more description about the AAA Transceiver thread and the SIP INFO message.

Version This manual is written to support the VOCAL Prepaid B2BUA Version 0.1.0.

Support The primary location for support, information and assistance for the VOCAL system is www.vovida.org. This site contains other documentation, training materials, development tools, development resources and informational mailing lists.

Preface

Introduction

This manual provides high-level design information for developers who are interested in working with the VOCAL B2BUA and its source code.

Objectives of this manual

This manual is intended as a technical briefing for C++ developers.

Intended audience

Software developers, architects, product managers and anyone else who is interest in the specific details about the function and coding of the VOCAL B2BUA.

Organization

This guide is organized as follows:

Chapter	***Title***	***Description***
Chapter 1	Specification	A high level overview of the system architecture.

Documentation conventions

The following is a list of conventions used in this guide:

Convention	***Description***
bold text	Names of elements found on the GUI screen, including buttons, and selectable entities such as, servers and server groups.
< >	Text that appears between angle brackets describes variables such as, <group name>.
`courier font`	System responses and prompts either from the CLI or GUI.

Convention	*Description*
`bold courier font`	Indicates information that you must enter.
■**Note**	Highlights points of additional interest for the user.
⚠ **Caution**	Be careful, this symbol highlights a potential for equipment damage or loss of data.

Terms found within this document

This section defines words, acronyms, and actions which may not be readily understood.

Table 1-1. Definitions

Term	*Definition*
SIP	Session Initiation Protocol
session	SIP session
transaction	SIP transaction
call leg	SIP call leg
B2BUA	An application that acts as a User Agent Server when it receives a SIP session invitation. It may act as a User Agent Client and create another call leg to the destination.
multi-leg call	A B2BUA call with one or more call legs.
VOCAL	Vovida Open Communication Application Library
RADIUS	Remote Authentication Dial In User Service
AAA	Authentication, Authorization and Accounting

Additional resources

On-Line Resources

Vovida.org (http://www.vovida.org) is a community web site dedicated to providing a forum for open source software used in datacom and telecom environment. This site was created to provide an environment where open source communications information and software can be easily located, accessed, retrieved and shared.

Table of Contents

Table of Contents *(continued)*

1

Specification

Topic	See Page

Problem Definition

Purpose

This project requires a Session Initiation Protocol (SIP: RFC 2543) application that can handle prepaid calls while being able to modify the media and initiate session tear-downs during established calls. This functionality is beyond the ability of standard SIP proxy servers, therefore we have implemented these functions into a Back-to-Back User Agent (B2BUA).

The B2BUA is defined in the draft (http://www.cs.columbia.edu/~hgs/sip/drafts/draft-ietf-sip-rfc2543bis-04.txt) as follows:

> Back-To-Back User Agent: Also known as a B2BUA, this is a logical entity that receives an invitation, and acts as a UAS to process it. In order to determine how the request should be answered, it acts as a UAC and initiates a call outwards. Unlike a proxy server, it maintains complete call state and must participate in all requests for a call. Since it is purely a concatenation of other logical functions, no explicit definitions are needed for its behavior.

This B2BUA acts as a User Agent Server (UAS) in the caller initiated call leg and creates another call leg to the destination as a User Agent Client (UAC). After the call is set up, the B2BUA may send SIP messages to modify the caller’s media to convey call duration or billing related information. It may also use Hypertext Transfer Protocol (HTTP: RFC 2616) messages to carry the same information. The B2BUA can control the call by tearing it down when the caller’s prepaid time has expired.

Stand alone

This application is a “stand alone” VOCAL SIP server: it has been built with Vovida Open Communications Applications Library (VOCAL) software components, but does not depend on other VOCAL servers such as, the Provisioning server. However, it is capable of working with other componentized VOCAL servers. Componentizing the VOCAL servers is a planned, future project.

User account information

User account information, such as prepaid amount or duration, is assumed to be stored persistently in an external Billing Server. The B2BUA uses Remote Authentication Dial In User Service (RADIUS: RFC 2138) messages to query and update user account balances. Other protocols, such as Open Settlement Protocol (OSP: A European Telecommunications Standards Institute (ETSI) protocol http://www.etsi.org/) messages could perform the same tasks, but this release of the B2BUA works only with RADIUS.

Media handling

In addition to handling SIP signaling, it would be desirable to handle media within the server. However, this would add latency to the media stream and limit the number of concurrent sessions that it can handle. The number of concurrent media sessions is undetermined, but the *guesstimate* is less than 50 G.711μ-law voice calls using a single high-end Pentium III processor and 1GB of memory.

■ **Note**
50 sessions mean 25 two-leg calls.

The initial implementation does not handle media locally in the server.

Design

Component reuse

The B2BUA has been mostly constructed from VOCAL components. Some new components have been added to the library for future re-use including the following:

- New RADIUS stack code
- AAA Transceiver
- UAC and UAS State Machines

Some existing components may be modified for inclusion within the stand-alone server. For example, PSLib will be able to access local files (or database(?)) instead of querying them from the existing VOCAL Provisioning server. Design and implementation should not prevent it from working as part of a VOCAL softswitch. Initially, there are no plans to test the B2BUA as a VOCAL server.

VOCAL RADIUS stack

The current VOCAL RADIUS stack code has been used for the VOCAL proxy server (Marshal) and Call Details Record (CDR) server. Instead of modifying it extensively to support this application, a new modular Authentication, Authorization and Accounting (AAA) client has been implemented to permit future re-use of the code. In addition, the AAA messaging protocol has been designed as another module to permit any future requirement to replace RADIUS with another protocol.

RADIUS servers from different vendors are likely to have different requirements for messages and attributes from their clients. Therefore, it is desirable to have a RADIUS client implementation that allows customization of its messages and attributes without rebuilding the object code.

HTTP server

The B2BUA may contain an HTTP server component that was developed by the Mascarpone (new Provisioning) project. HTTP clients at the end users or other servers may use HTTP requests to retrieve or update billing information. It can also be used in the future as an interface for third party call control. The initial phase of this project does not include this component.

bis-04 draft compliant

The B2BUA needs to be SIP bis-04 draft compliant in the final release. Therefore, some code in the current SIP stack needs to be changed.

A reusable design

It is desirable to have a reusable design for future implementation of a Third Party Call Controller, Voice XML Server, Middlebox Communication (MIDCOM) Agent, Multi-line User Agent or other applications. Each call leg has its own UAS or UAC state machine and communicates with the multi-leg call control through a FIFO queue and a third party call control type interface.

Overview

Call initiation

When an INVITE comes into the B2BUA, it queries the Billing Server for the caller's unused prepaid call time via a RADIUS Accounting Start message. If there is no more unused time left, it returns a “402 Payment Required” (or “403 Forbidden” because 402 is indicated as Reserved for future use).

It may also need to check if the caller has at least a minimal amount of remaining prepaid time, for example, 1 minute. The actual RADIUS messaging sequence for prepaid probably varies among different billing server implementations. RADIUS Access-Request messages can be used to obtain user authorizations from the RADIUS server. Additional Attributes or Vendor-Specific Attributes (VSAs) may need to be added to meet a specific billing server's requirements.

Call establishment

If there is unused time, the Multi-Leg Call Control (MLCC) initiates another call leg to the callee. Once the callee answers the call, it sends a RADIUS Accounting Interim message to the billing server to record the fact that the call has been answered. It may also send an INFO message upstream to the caller with the user's account balance. In addition, it may start a timer with the duration of a predefined number of seconds (180?) or unused time, whichever is less.

When the timer expires, the B2BUA sends a RADIUS Accounting Interim to the Billing Server and deducts the amount from the balance locally. If the balance is below a predefined duration, for example, 30 seconds, the B2BUA may warn the caller by sending an INFO message upstream and/or inject an additional compatible media stream into the caller's media stream. The source of the media may be a SIP or Real Time Streaming Protocol (RTSP: RFC 2326) Media Server.

■ **Note**

Working with a media server will not be implemented in the initial release.

These checkpoints allow a single account to make long duration calls. However, if it is not possible with the Billing Server or the currently defined RADIUS messages, the B2BUA will update the Billing Server only at the termination of a call with an Accounting Stop message.

Checking remaining prepaid time

If there is more time left, the B2BUA starts the timer again with the same operations as described above. If there is no more time left, the call legs will be disconnected by sending BYE messages to all call legs and a RADIUS Accounting Stop message to the Billing Server, instead of the Accounting Interim message, to terminate the call. If either side disconnects the call, the B2BUA stops accounting with the elapsed time since the previous update.

HTTP requests

(The HTTP operations are as of yet, unknown.) If somehow the B2BUA is able to associate an HTTP request with a multi-leg call (which one came first?), it should be able to send responses to the client upon call control events.

Source code

The B2BUA uses most of the SIP proxy base code. It extends CommandLine to support stand-alone mode. The Provisioning Server address and port number arguments become optional in this mode. By default, PSLib looks for the configuration file b2bua.xml in the directory where the application is started. If the Provisioning Server address is provided, it retrieves a configuration file from there. In addition, the application becomes a “HeartlessProxy” instead of a “BasicProxy” in stand-alone mode.

A HeartlessProxy is one that does not multicast heartbeating pulses. Heartbeating is used within VOCAL’s network management scheme.

Threads

Illustration

Figure 1-1 shows how different B2BUA threads communicate through FIFO queues when they are running.

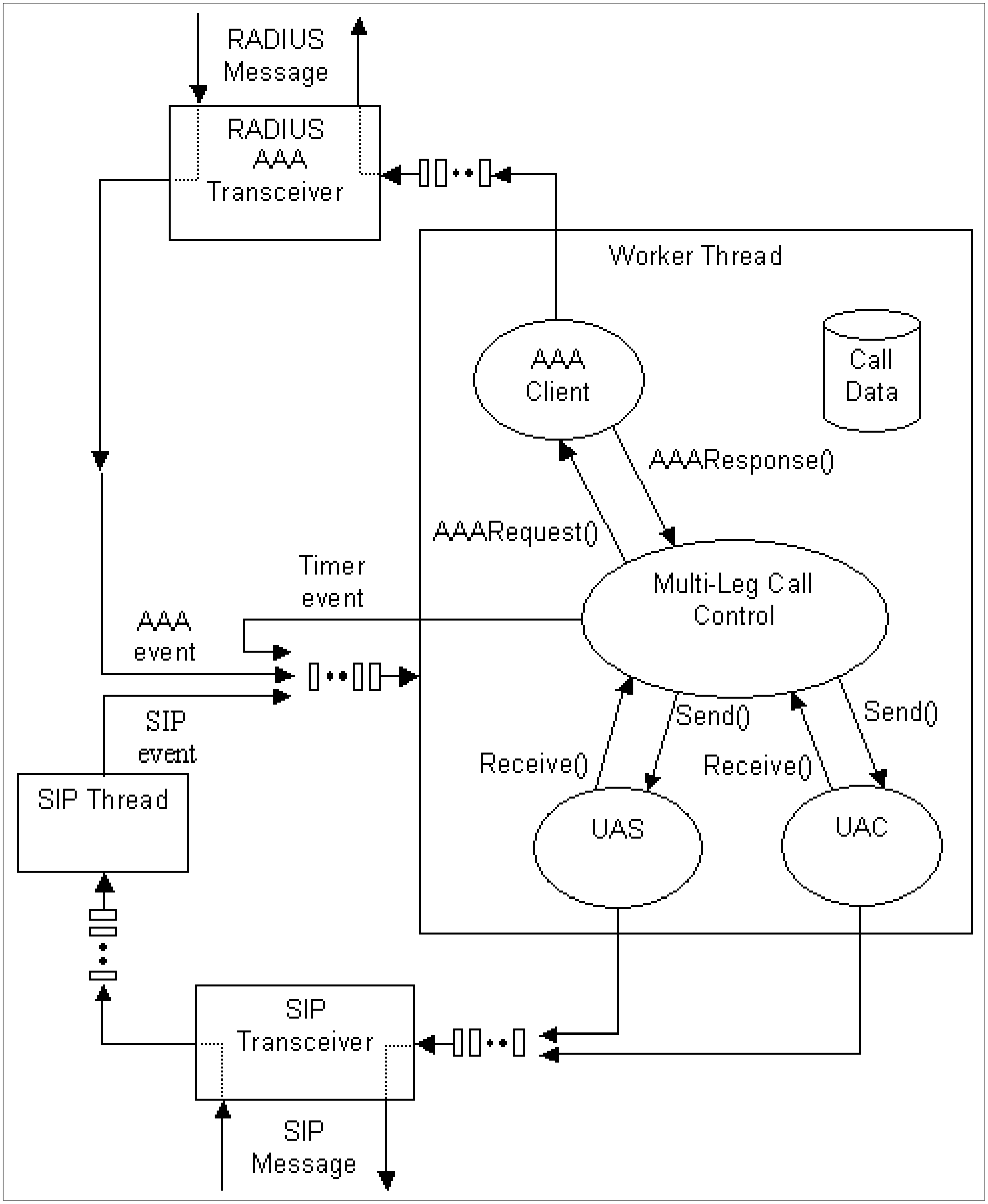

Figure 1-1. Figure 1. B2BUA Threads.

SIP thread and transceiver

The SIP thread and Transceiver are the same as the ones used in the other existing applications. The other existing SIP proxy base code threads are not shown because there are no major changes and they are not interesting in stand-alone mode.

Worker Thread

State machines

The Worker thread contains the state machines for MLCC, UAS and UAC. These state machines use the shared Multi-Leg Call Data (MLCD) and run within the same thread so that it is not necessary to lock the call data. Events in the Worker Thread FIFO queue can be either SIP messages, AAA responses or time-outs.

SIP INVITE event

If the event is a SIP INVITE message and the call leg does not exist in the call database, it is processed by the MLCC state machine as a new call. It sends a SIP “100 Trying” message upstream and creates a new MLCD object with one UAS call leg. It then creates an accounting request event and adds it to the Accounting RADIUS Client Thread’s FIFO queue.

MLCC sends call-leg control events back to the FIFO queue of it own thread for the UA state machines to use. All other non-SIP and non-call-leg control events are also handled by this state machine. One or more call-legs with UAC state machines may be created for future releases.

Other SIP events

If the event is a SIP message of an existing call-leg or a call-leg control event, it is handled by the call leg’s state machine. It can either be a UAS or UAC state machine. The UA state machines communicate with MLCC by composing and sending B2B call control events back to the Worker Thread’s FIFO queue. Call-legs of a multi-leg call can communicate with each other via call-leg control events through the queue.

RADIUS AAA Transceiver Thread

Purpose

The purpose of the Transceiver is to separate the functionality of the application from being dependent upon a specific AAA protocol. If in the future, for example, it is determined that COPS would serve the accounting function better than RADIUS, a COPS AAA Transceiver thread could be written to enable the application to work with COPS instead of or in addition to RADIUS.

Translating Messages

The AAA Transceiver sends and receives RADIUS messages and is the interface that any application would use to communicate with the RADIUS server. This AAA Transceiver encapsulates everything that RADIUS needs: it takes a request from the application, in the application’s format, and translates it into a RADIUS request and sends it to back the RADIUS server. When it receives a response from the server, it translates the response into a format that the application can understand, and then forwards the response to the application.

Accounting On/Off

Some servers require that the Accounting RADIUS Client Thread send Accounting On during start up and Accounting Off during shutdown along with Keep Alive message while it is running. While the RADIUS stack is capable of transmitting these message, this ability has not been implemented into the AAA Tranceiver Thread.

Call Accounting

As for the call, an intermediate accounting message is sent once the caller picks up the phone. In future releases, a multiple of intermediate messages will be sent at a configured frequency to avoid free calls, should the B2BUA go down.

Authorization Requests

When accounting is “on”, it listens for authorization requests from MLCC. Each authorization request is assigned to a thread from a thread pool that can be tuned to the input load. Each thread communicates with the Billing Server via RADIUS messages. When it receives the accounting response, it creates an AAA event and adds it to the Worker Thread FIFO queue.

HTTP Server Thread

Not implemented.

Call Data

Illustration Figure 1-2 shows the relationship among various call data.

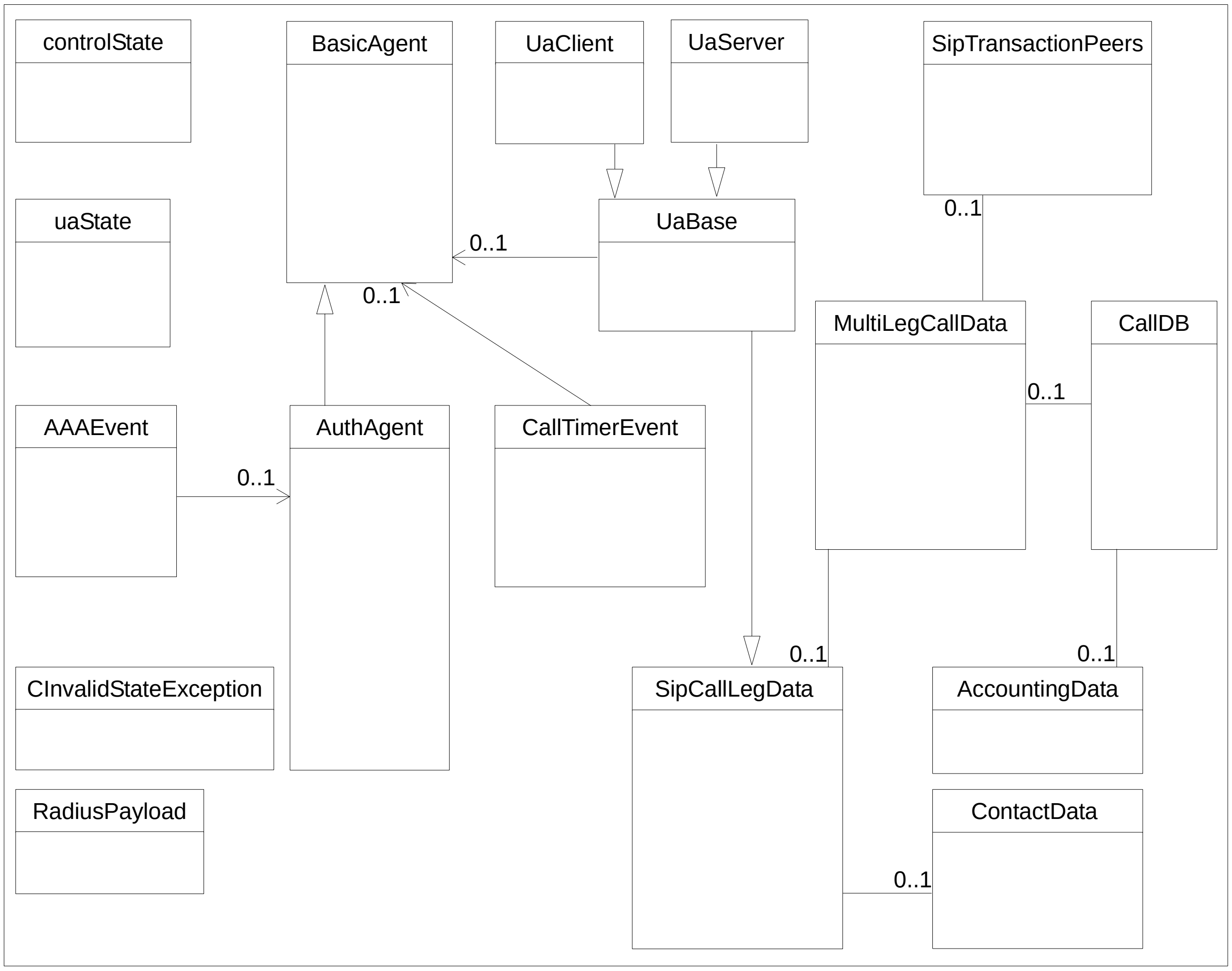

Figure 1-2. Call Data Class Diagram

Class descriptions Figure 1-1 describes the classes found within the B2BUA.

Table 1-1. Class Descriptions

Class Name	*Description*
AAAEvent	Processes Authentication, Authorization and Accounting events.

Table 1-1. Class Descriptions (Continued)

Class Name	***Description***
AccountingData	This is a map of Accounting Session Id and smart pointer to Multi-Leg Call Data pairs. It is used for finding the multi-leg call when an accounting response is received. An Accounting Session Id is a unique string for identifying a query to the Billing Server. It may be the SIP CallId of the UAS call leg.
AuthAgent	Authorizes Authentication, Authorization and Accounting clients.
BasicAgent	Base proxy class.
CallDB	This is a map of SIP Call Leg and smart pointer to Multi-Leg Call Data pairs. It is used for finding the multi-leg call when a SIP message event is received.
CallTimerEvent	A timer that is configured to expire like an alarm.
CInvalidStateException	A C++ programming aid that indicates errors as they are propagated up through the application.
ContactData	This class contains these attributes: • Remote SDP - Remote SDP of this contact • Peer List - a Peer Table of other call legs involved
ControlState	Base proxy class.
MultiLegCallData	Multiple entries in the Call Legs Container and Accounting Session Id Container may point to a common MLCD. Attributes: • Call State - MLCC state machine states • A smart pointer to FSM • Call Legs - A vector of smart pointers to SIP Call Leg Data (or a map of Call Leg and smart pointers to SIP Call Leg Data pairs) • SIP Transaction Peers - A map of SIP transaction Id and Peer Table pairs. Not all transactions involve all call legs. Accounting data - Accounting Session Id, unused credit, call start time, elapsed time, etc. (expand this to a separate object?)
RadiusPayload	The RADIUS message packet.

Table 1-1. Class Descriptions (Continued)

Class Name	***Description***
SipCallLegData	Attributes • Call Leg - It's own Id (not needed if Multi-Leg Call Data use map instead of vector) • State - Active or Inactive (on hold) • Call State - UA state machine states • A smart pointer to FSM • Contact List - A vector of pointers to Contact Data with the last one being current • Route - SIP Route header for subsequent requests
SipTransactionPeers	This is a type definition of a vector of smart pointers to Sip Call Legs.
UaBase	Base proxy class.
UaClient	Base proxy class.
UaServer	Base proxy class.
uaState	Base proxy class.

Finite State Machines

No replication

State machines do not store any call related data locally so they don't need to be replicated for each call.

TODO

Make sure they handle re-INVITE, proxy unknown methods, reject unsupported methods, for example.

Multi-Leg Call Control Finite State Machine

Illustration

Figure 1-3 illustrates the Multi-Leg Call Control State Machine.

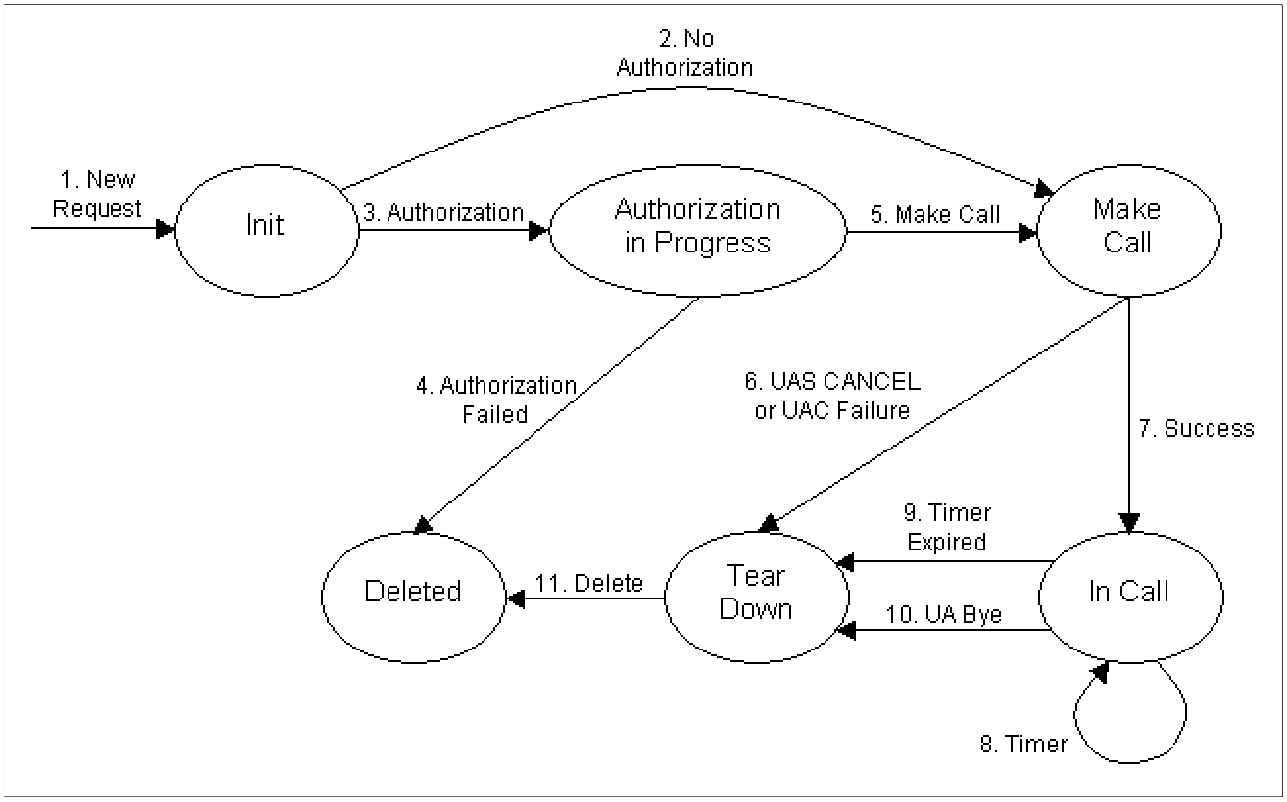

Figure 1-3. Multi-Leg Call Control State Machine

Operation

This is the state machine for the authorization process. Based on the type of application that is being used, there is an agent assigned, or created, for each service and each agent goes through its own state machine. For example, for prepaid billing, there is an Auth agent object, which goes through the different states shown in Figure 1-3.

transitions

Table 1-2 describes the transitions within the state machine as illustrated in Figure 1-3.

■**Note**

This state machine is intended only for pre-paid billing. Other applications, such as click-to-dial, will require other state machines.

The operator numbering is intended as labeling for clear identification: There is no requirement for following all transitions in order. For example, if there is no authorization, the object will take step 2 bypassing transitions 3, 4 and 5.

In this implementation of the B2BUA, all calls are authorized through the RADIUS server. The No Authorization step has been made available for future development with new applications that do not require authorization.

Table 1-2. MLCC State Machine Description

Transitions	***Description***
1) New Request	When a request is received, such as a SIP INVITE message, an object is created, taking the state machine to the Init state.
2) No Authorization	If the object bypasses authorization, it bypasses the Authorization in Progress state and takes the state machine to the Make Call state.
3) Authorization	If authorization is required, a request is sent to the AAA Transceiver for the authorization, and takes the state machine to the Authorization in Progress state. This is asynchronous: The AAA Transceiver will talk to the RADIUS server, which will take its own time to respond.
4) Authorization Failed	If the authorization fails, the state machine is taken to the Deleted state where the object is sent to garbage collection and the process ends.
5) Make Call	If the authorization succeeds, the state machine is taken to the Make Call state and an attempt is made to connect the two ends together. The INVITE, which had been held back pending authorization, is sent to its intended destination.
6) UAS CANCEL or UAC Failure	If the user hangs up before the call is set up or the client receives a failure response (404, for example) the state machine is taken to the Tear Down state.
7) Success	If the far end responds favorably to the INVITE message (180, 200), the call is successfully connected, which takes the state machine to the In Call state. At this point a SIP INFO message is sent to the caller stating the time remaining in the call.

Table 1-2. MLCC State Machine Description (Continued)

Transitions	*Description*
8) Timer	The two parties are engaged in a call. The RADIUS server, as part of its authorization response, queries the calling party's credit, and starts a timer to count down the remaining time left in the account.
9) Timer Expired	If the timer expires before the users hang up, the RADIUS server signals for the call to end, and the B2BUA sends a mandatory BYE to both sides of the call taking the state machine to the Tear Down state.
10) UA BYE	If one of the users hangs up before the timer expires, a BYE is sent and the state machine is taken to the Tear Down state.
11) Delete	The object is sent to garbage collection and the process ends.

User Agent Server Finite State Machine

Illustration

Figure 1-4 illustrates the User Agent Server State Machine.

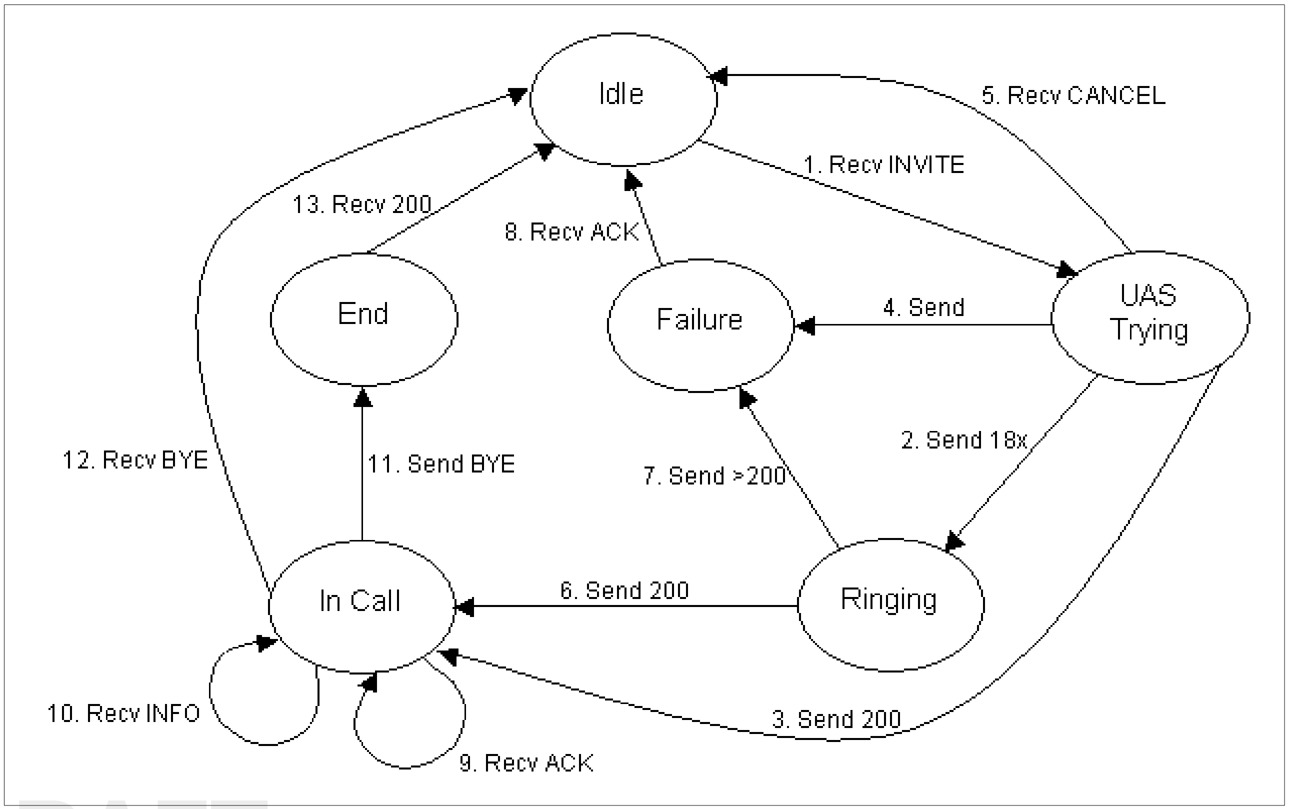

Figure 1-4. User Agent Server State Machine

Responsibilities

The UAS state machine is responsible for the transitions between states at the callee side of a call leg. It handles messages received from the SIP stack and sends SIP message requests from MLCC. The SIP stack keeps track of the SIP transaction and retransmission, thereby relieving this state machine from that duty. The initial state in the SIP Call Leg data is Idle when it is associated with this state machine.

Transitions

The call state will change over time according to the transitions described in Table 1-3.

Table 1-3. User Agent Server State Machine Transitions

Transitions	*Description*
1) Recv INVITE	MLCC receives an INVITE for a new session, creates this call leg and tries to contact the callee. The next state is UAS Trying.
2) Send 18x	The callee alerts: MLCC instructs the UAS to send a 18x message to the caller. The next state is Ringing.
3) Send 200 (INVITE)	The callee answers immediately without going through the alerting phase. MLCC instructs the UAS to send a 200 OK message to the caller. The next state is In Call.
4) Send >200	The session cannot be established, so MLCC instructs the UAS to send the corresponding failure code to the caller. The next state is Failure.
5) Recv CANCEL	The UAS receives a CANCEL message from the caller and sends back a 200 OK response, and notifies MLCC of the cancellation. The next state is Idle.
6) Send 200 (INVITE)	The callee answers the call. MLCC instructs the UAS to send a 200 OK message to the caller. The next state is In Call.
7) Send >200	See Transition 4.
8) Recv ACK	The UAS receives an ACK from the caller for the failure status message. It notifies MLCC of the transaction’s completion. The next state is Idle.
9) Recv ACK	The UAS receives an ACK from the caller for the 200 OK status message. It notifies MLCC of the transaction’s completion.
10) Recv INFO	The UAS receives an INFO message from the caller, sends back a 200 OK response, and notifies MLCC of the INFO message.
11) Send BYE	The MLCC instructs the UAS to terminate the call by sending a BYE message to the caller. The next state is End.

Table 1-3. User Agent Server State Machine Transitions

Transitions	*Description*
12) Recv BYE	The UAS receives a BYE message from the caller, sends back a 200 OK response, and notifies MLCC of the termination.The next state is Idle.
13) Recv 200 (BYE)	The UAS receives a 200 OK response for the BYE. It notifies the MLCC of the transaction’s completion. The next state is Idle.

User Agent Client Finite State Machine

Illustration

Figure 1-5 illustrates the User Agent Client State Machine.

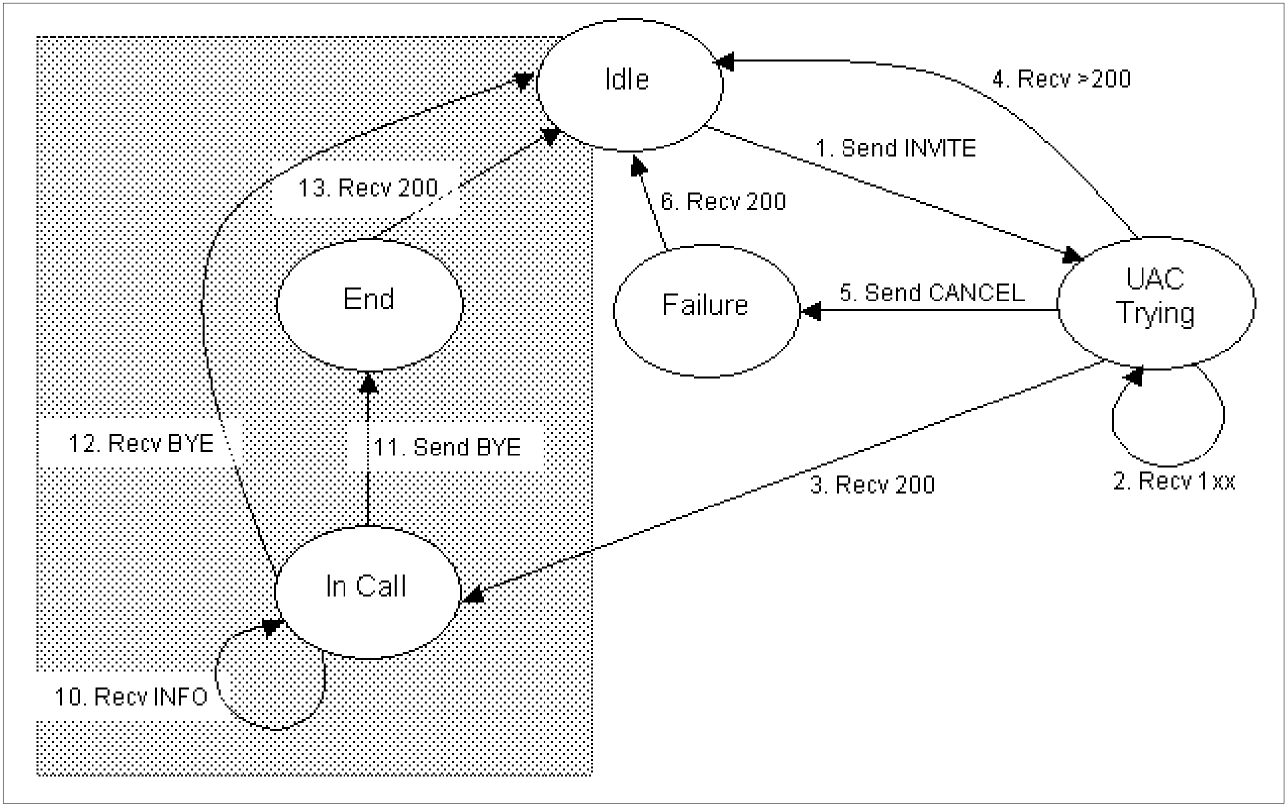

Figure 1-5. User Agent Client State Machine

Responsibilities

The UAC state machine is responsible for the transitions between states at the caller side of a call leg. It handles messages received from the SIP stack and sends SIP message requests from MLCC. The SIP stack keeps track of the SIP transaction and retransmission, thereby relieving this state machine from that duty. The initial state in the SIP Call Leg data is Idle when it is associated with this state machine.

Transitions

The call state will change over time according to the transitions described in Table 1-4.

■**Note**
The transitions, 10 to 13, in the shaded area of Figure 1-5 are identical to those of the UAS FSM as shown in Figure 1-4, and therefore their descriptions are not included in Table 1-4.

Table 1-4. User Agent Client State Machine Transitions

Transition	***Description***
1) Send INVITE	MLCC instructs the UAC to initiate a session to the callee. The next state is UAC Trying.
2) Recv 1xx	The UAC receives a 1xx provisional message from the callee and notifies the MLCC of the status message. The call state is not changed.
3) Recv 200 (INVITE)	The UAC receives a 200 OK response for the INVITE message from the callee. It notifies MLCC of the callee answering the call. ■**Note** An ACK is not sent yet since the SDP information may be changed by the MLCC. The next state is In Call.
4) Recv >200	The UAC receives a failure final response from the callee. It Sends an ACK to the callee to complete the transaction and Notifies MLCC of the failure.The next state is Idle.
5) Send CANCEL	MLCC wants to abort the session and instructs UAC to send a CANCEL to the callee. The next state is Failure.
6) Recv 200 (INVITE)	The UAC receives a 200 OK response for the CANCEL and notifies MLCC of the transaction's completion. The next state is Idle.
7) Send ACK	MLCC instructs the UAC to send an ACK to the callee to complete the transaction. The call state is not changed.

In-Call State Re-INVITE Handling

Illustration

Figure 1-6 illustrates the in-call state transitions for a Re-INVITE message.

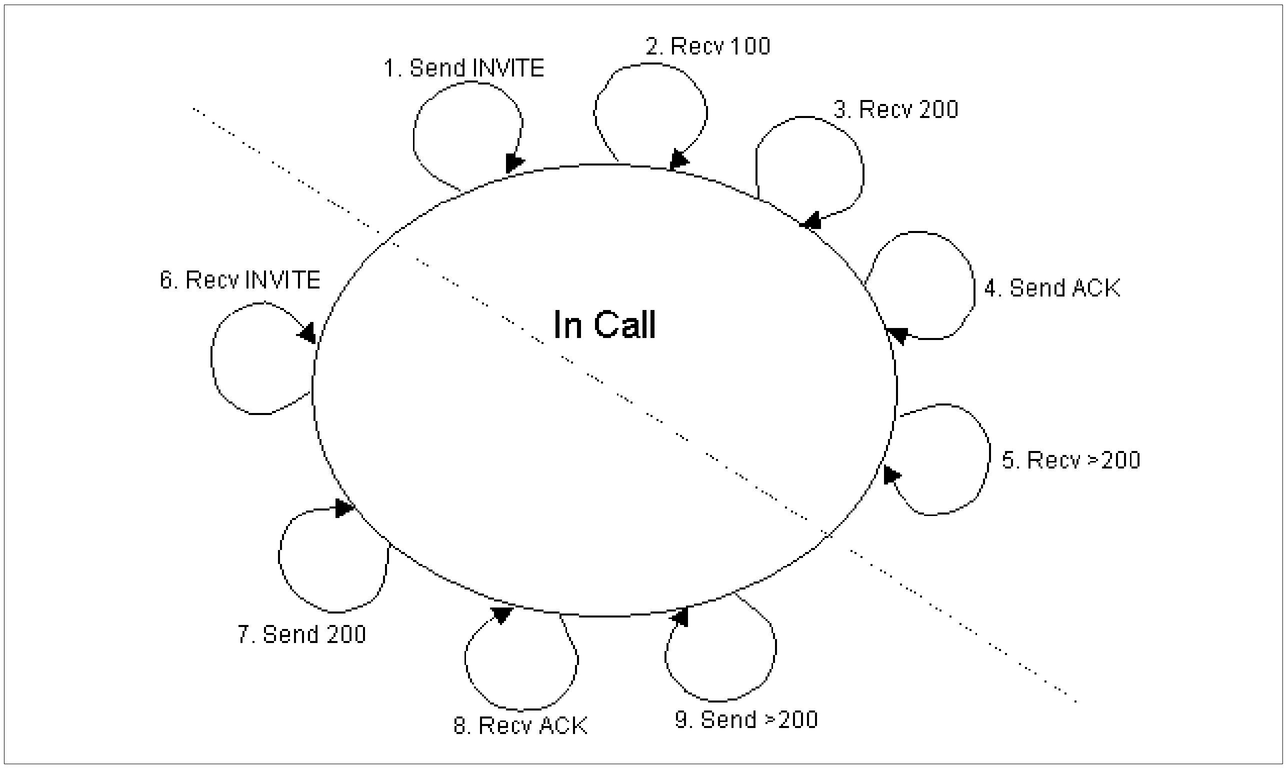

Figure 1-6. In-Call state transitions for Re-INVITE.

Common to UAC and UAS

This section is common to both the UAC and UAS.

- Self-transitions 1 to 5 are for a re-INVITE transaction initiated from MLCC.
- Transitions 6 to 9 are for a re-INVITE transaction initiated from the remote User Agent.

In general, all “Send xx” transitions are instructed by MLCC and all “Recv xx” transitions are triggered by reception of SIP messages from the remote UA. MLCC is notified of all “Recv xx” events. The call leg is in In Call state, therefore, if the re-INVITE request fails (Recv >200 or Send >200), the Session Description Protocol (SDP) agreed upon in the last successful INVITE transaction remains in force. Therefore, the call state is not changed for any of these transitions.

Memory and Performance Impact

Overview

Details such as, calls per second, are TBD.

The B2BUA is slower and bigger than a basic proxy due to additional message parsing, RADIUS queries, HTTP connections, call legs lookup, call control, for example. If the media is handled locally, it will be much slower than it already is. In addition, media quality will be worse with the additional latency expended by forwarding media packets.

End User Interface

Section contents

This section contains information about the following:

- Caller and Callee
- AAA

SIP INFO Message

Purpose

The SIP INFO message is used to inform the user about how much pre-paid credit remains in his or her account.

Message Content

This is an example of an INFO message sent from the B2BUA when the caller picks up the phone. This message is sent just after the Interim account record.

```
INFO sip:128.107.140.141:5084 SIP/2.0
Via: SIP/2.0/UDP 128.107.140.141:5065
From: 2000<sip:2000@128.107.140.141:5084;user=phone>
To: 1001<sip:1001@128.107.140.141:5065;user=phone>
Content-Length: 34

You have (20) seconds for the call
```

In the future, a warning will be sent to the user indicating that only 1 minute of credit remains.

AAA Messages

Interface

In version 0.1.0 of the B2BUA, the Authentication, Authorization and Accounting (AAA) server is a RADIUS-based billing server. In future releases, other protocols, such as COPS, could be used to provide AAA.

RADIUS Messages

Overview

The billing server provides access and accounting control through RADIUS messages. This section gives a brief overview of these messages.

Access

Definition

RADIUS Access messages are literally requests for authorization that may be accepted or rejected based on the provisioned billing information.

Access-Requests

An Access-Request message contains a unique user ID, which is sent to the RADIUS server for authorization.

Access-Responses

The responses are either Access-Accept or Access-Reject.

Access-Accept

The Access-Accept message contains information about how much credit the user has left in his or her account. If the prepaid time-remaining is 0, the call is dropped immediately after notifying the user through an INFO message. However, the billing server response is still an Access-Accept message.

Access-Reject

The most common causes for an Access-Reject message are the absence of an account or a lack of pre-paid time remaining in the user's account. If there is a problem with either the Billing server or the communication link between the user and the server, the client tries three times to send the Access-Request at 3-second intervals. If the response does not come back from billing server it is treated as an Access-Reject.

In future releases, there will be a more verbose message as to why the rejection is happening.

Accounting

Definition

Once the user has been authorized, and the call is set up, the accounting messages work with the billing server to charge the user's account for the length of the call.

Accounting-Requests

Start

The Start message contains the calling party's user ID and a timestamp indicating the start of the call.

Interim

When the called party picks up the phone, an Interim message is sent. The interim time is the ring time: the elapsed time between when the phone started ringing and the called party picked it up. It is calculated from the difference between the timestamps for the Start time and the Interim time. In some cases, this ring time is billed to the caller.

Stop

When the call ends, a Stop message is sent. The difference between the timestamp for the Interim time and that of the Stop time equals the call time.

The difference between the Start and Stop times equals the entire use of the resources.

Accounting-Responses

These tell the Transceiver that the requests have been received and to stop retransmission. Otherwise, the content of a response is ignored.

Configuration and Restrictions

Issues

Configuration data: local SIP port, Billing Server (address, port, authentication, password), local RADIUS port, call check point duration (180seconds?), warning duration(s) (30 seconds and 20 seconds?), thread pool size, UAC side proxy server address, HTTP Server port, redundant B2BUA (TBD) TODO: include sample b2bua.xml

- No registration
- Do not support REFER, SUBSCRIBE/NOTIFY
- No user authentication locally
- No media handling in initial release, but may consider working with media server if one available and time permits.
- First release will not work in VOCAL.
- No Session timer support
- No Redundancy

Testing Considerations

Tested as a stand-alone server

The B2BUA will be tested as a stand-alone server. If we cannot find a UAC that can handle mid-call media changes and SIP INFO messages from the B2BUA, the existing VOCAL UA will be modified for testing. We also need to write a HTTP client test driver that sends HTTP requests for billing information. This may as well be the VOCAL UA.

- Test drivers that simulate a RADIUS Server will be developed. Test with actual Billing server if available.
- No testing with media server.
- Automate and document test cases with “make test” and Verify utility.

Design Specifications for Reliability and Availability

Not in release	TBD. It is not considered in the first release.

Reference Documents

URLs

Figure 1-5 lists the URLs for some reference documents.

Table 1-5. Reference Document URLs

Document	***URL***
RFC2543bis (-04) draft	http://www.cs.columbia.edu/~hgs/sip/drafts/draft-ietf-sip-rfc2543bis-04.txt
RFC2865	http://www.ietf.org/rfc/rfc2865.txt
RFC2866	http://www.ietf.org/rfc/rfc2866.txt
MIDCOM framework draft	http://www.ietf.org/internet-drafts/draft-ietf-midcom-framework-03.txt

Index

Index (Continued)

Index (Continued)