

VOCAL

Vovida Open Communication Application Library

VOCAL Prepaid B2BUA

Software Version 0.1.0 (Alpha)

DRAFT

CopyrightCopyright © 2001, Cisco Systems, Inc.

Revision History

The following table itemizes the revision history of this manual:

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

Table 1. Software Release 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

Table 2. Documentation History

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

SupportThe 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.

DRAFT

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:

<i>Chapter</i>	<i>Title</i>	<i>Description</i>
Chapter 1	Specification	A high level overview of the system architecture.

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

<i>Convention</i>	<i>Description</i>
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>.
<code>courier font</code>	System responses and prompts either from the CLI or GUI.

DRAFT

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.

DRAFT

Table of Contents

Preface	iii
Chapter 1.	
Specification	
Problem Definition	1-2
Design	1-3
Memory and Performance Impact	1-17
End User Interface	1-18
AAA	1-20
Configuration and Restrictions	1-22
Testing Considerations	1-23
Design Specifications for Reliability and Availability	1-24
Reference Documents	1-25

Table of Contents *(continued)*

Specification

Topic	See Page
Problem Definition	1-2
Design	1-3
Overview	1-4
Threads	1-6
Worker Thread	1-7
RADIUS AAA Transceiver Thread	1-7
HTTP Server Thread	1-7
Call Data	1-8
Finite State Machines	1-11
Multi-Leg Call Control Finite State Machine	1-11
User Agent Server Finite State Machine	1-12
User Agent Client Finite State Machine	1-14
In-Call State Re-INVITE Handling	1-16
Memory and Performance Impact	1-17
End User Interface	1-18
Caller and Callee	1-19
AAA	1-20
Requests to RADIUS Server	1-20
Access-Request	1-20
Accounting-Request	1-20
Responses From RADIUS Server	1-20
Configuration and Restrictions	1-22
Testing Considerations	1-23
Design Specifications for Reliability and Availability	1-24
Reference Documents	1-25
Attachments	1-26

DRAFT

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.711 μ -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.

DRAFT

Design

Component reuse	<p>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:</p> <ul style="list-style-type: none"> • New RADIUS stack code • AAA Transceiver • UAC and UAS State Machines <p>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.</p>
VOCAL RADIUS stack	<p>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.</p> <p>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.</p>
HTTP server	<p>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.</p>
bis-04 draft compliant	<p>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.</p>
A reusable design	<p>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.</p>

DRAFT

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.

DRAFT

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.

DRAFT

Threads

Illustration

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

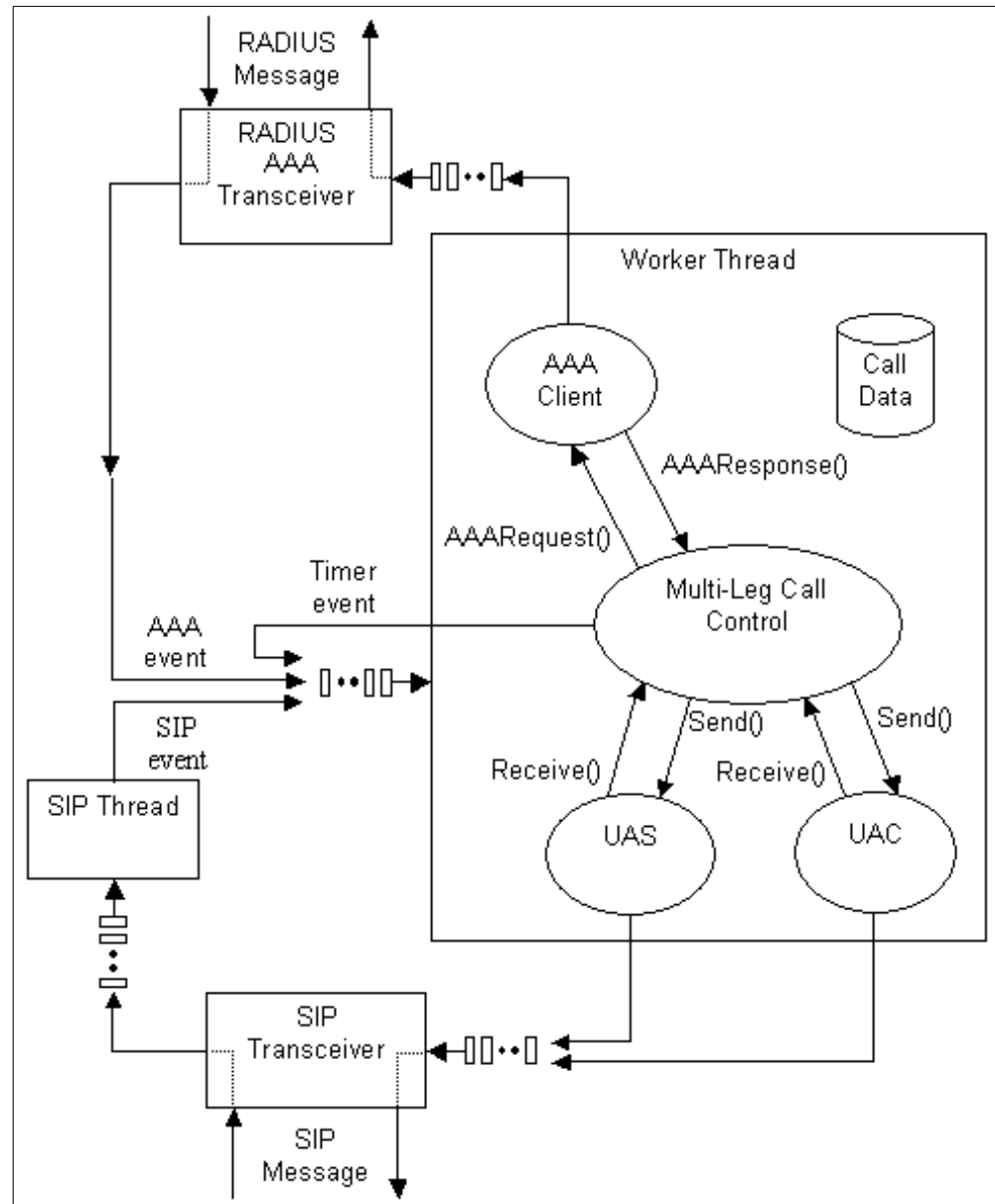


Figure 1-1. Figure 1. B2BUA Threads.

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.

DRAFT

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	<p>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.</p> <p>MLCC sends call-leg control events back to the FIFO queue of its 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.</p>
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

Sending RADIUS messages	<p>The AAA Transceiver sends and receives RADIUS messages. Add more details.</p> <p>(I am not sure about this.) The Accounting RADIUS Client Thread sends Accounting On during start up and Accounting Off during shutdown. It also sends Keep Alive messages periodically.</p>
Accounting	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.

DRAFT

Call Data

Illustration

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

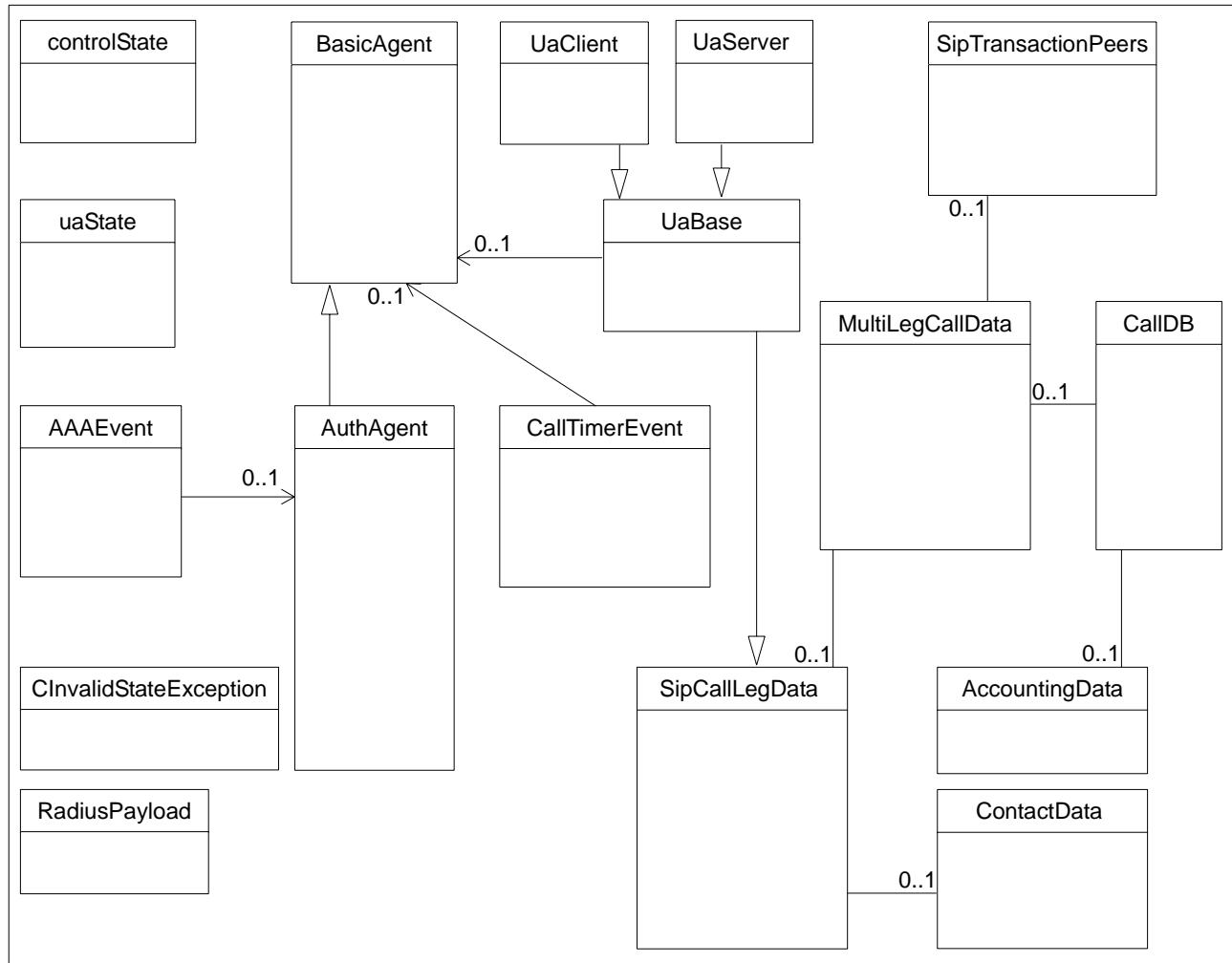


Figure 1-2. Call Data Class Diagram

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.

DRAFT

Table 1-1. Class Descriptions

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.
CallIDB	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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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.

DRAFT

Table 1-1. Class Descriptions

Class Name	Description
SipCallLegData	Attributes <ul style="list-style-type: none">• 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.

DRAFT

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, etc.

Multi-Leg Call Control Finite State Machine

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

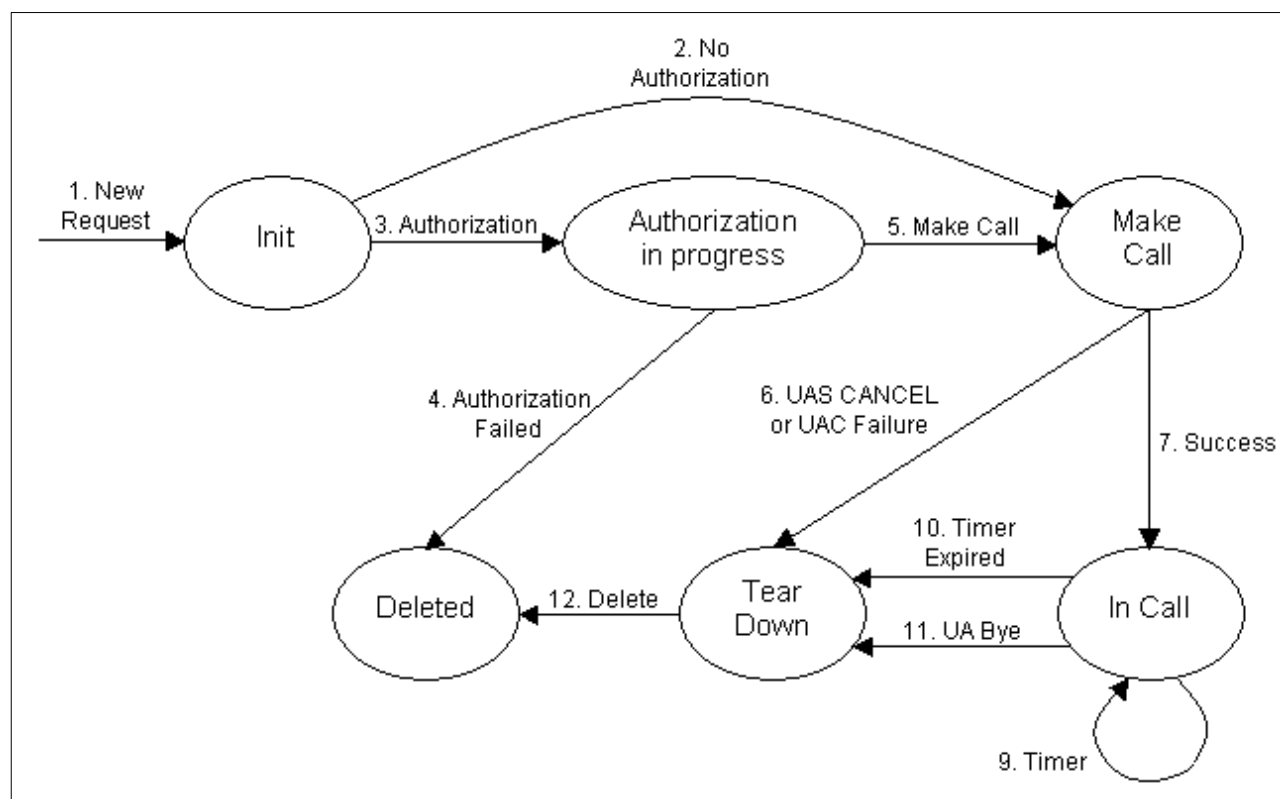


Figure 1-3. Multi-Leg Call Control State Machine

TODO Add descriptions of states, operators and events

DRAFT

User Agent Server Finite State Machine

Illustration

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

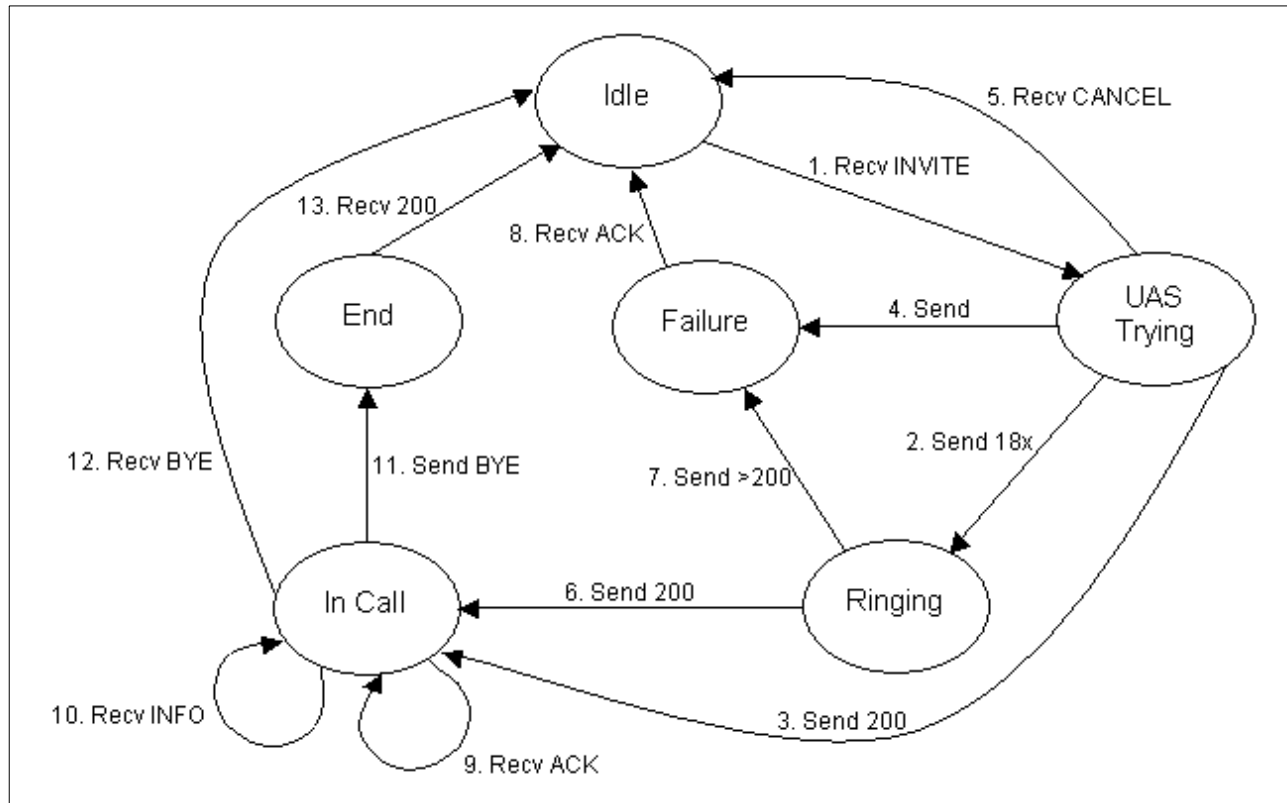


Figure 1-4. User Agent Server State Machine

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 1-2.

Table 1-2. 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.

DRAFT

Table 1-2. User Agent Server State Machine Transitions

Transitions	Description
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.
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.

DRAFT

User Agent Client Finite State Machine

Illustration

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

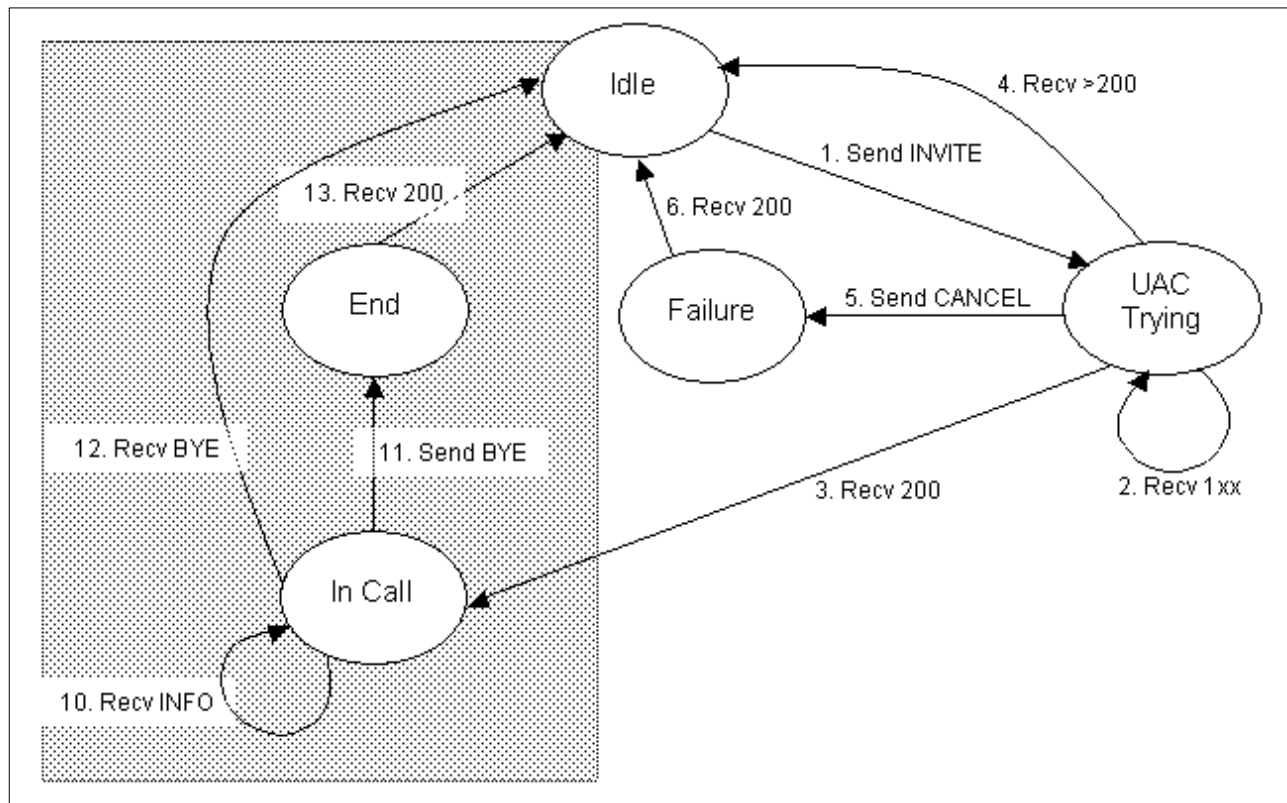


Figure 1-5. User Agent Client State Machine

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.

DRAFT

Transitions

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

■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 1-3.

Table 1-3. 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.

DRAFT

In-Call State Re-INVITE Handling

Illustration

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

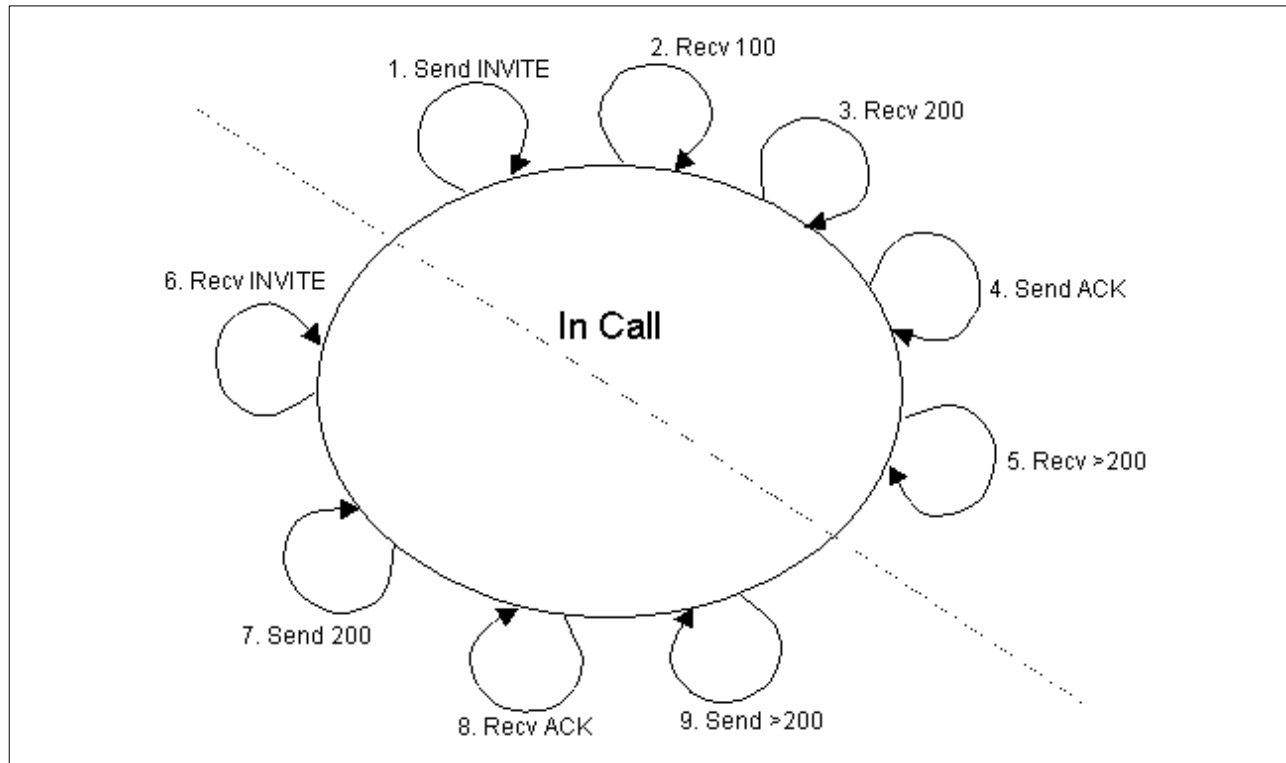


Figure 1-6. In-Call state transitions for Re-INVITE.

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.

DRAFT

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.

DRAFT

End User Interface

Section contents

This section contains information about the following:

- Caller and Callee
 - AAA
-

DRAFT

Caller and Callee

Interface

Interface to the caller and callee will be via messages.

TODO

Document the INFO message content here
Document the HTTP messages here when they are added later on.

DRAFT

AAA

Interface

Interface to the AAA (Billing) server will be RADIUS messages.

TODO

Document the RADIUS messages and attributes here

Requests to RADIUS Server

Overview

Access-Request

Description

Accounting-Request

Descriptions

Figure 1-4 describes the RADIUS accounting request messages.

Table 1-4. RADIUS Accounting Request Messages

<i>Request</i>	<i>Description</i>
Start	
Stop	
Interim	

Responses From RADIUS Server

Descriptions

Figure 1-5 describes the RADIUS responses.

Table 1-5. RADIUS Responses

<i>Response</i>	<i>Description</i>
Access-Accept	
Access-Reject	
Accounting-Response	

DRAFT

TODO

If RTSP messages are used, document them.
Interface to the media server will be SIP (or RTSP?) messages.

DRAFT

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
-

DRAFT

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.
-

DRAFT

Design Specifications for Reliability and Availability

Not in release

TBD. It is not considered in the first release.

DRAFT

Reference Documents

URLs

Figure 1-6 lists the URLs for some reference documents.

Table 1-6. 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

DRAFT

DRAFT

Index

Numerics

402 Payment Required 1-4
403 Forbidden 1-4

A

AAA
 see Authentication, Authorization and Accounting
AAAEvent 1-8
AccountingData 1-9
AuthAgent 1-9
Authentication, Authorization and Accounting
 events 1-7
 RADIUS stack 1-3
 worker thread 1-7

B

B2BUA
 see Back-to-Back User Agent
b2bua.xml
 Back-to-Back User Agent 1-5
 PSLib 1-5
Back-to-Back User Agent
 b2bua.xml 1-5
 defined 1-2
BasicAgent 1-9
Billing Server 1-2
bis-04
 see Session Initiation Protocol

C

call data
 illustration 1-8
call establishment 1-4
call initiation 1-4
CallDB 1-9
Caller and Callee 1-19
CallTimerEvent 1-9
CInvalidOperationException 1-9
classes
 AccountingData 1-9
 AuthAgent 1-9
 BasicAgent 1-9
 CallDB 1-9
 CallTimerEvent 1-9
 ContactData 1-9
 ControlState 1-9
 descriptions 1-8
 diagram 1-8
 MultiLegCallData 1-9
 RadiusPayload 1-9
 SipCallLegData 1-10
 SipTransactionPeers 1-10
 UaBase 1-10

UaClient 1-10
UaServer 1-10
uaState 1-10
component reuse
 discussion 1-3
 PSLib 1-3
componentizing 1-2
Configuration and Restrictions 1-22
ContactData 1-9
ControlState 1-9

D

Design 1-3
Design Specifications for Reliability and Availability 1-24

E

End User Interface 1-18
ETSI
 see European Telecommunications Standards Institute
European Telecommunications Standards Institute 1-2

F

Finite State Machines 1-11
functionality 1-2

G

G.711 1-2

H

heartbeating 1-5
HeartlessProxy 1-5
HTTP
 see Hypertext Transfer Protocol
Hypertext Transfer Protocol
 requests 1-4
 RFC 1-2
 server 1-3
 server thread 1-7

I

illustrations
 call data 1-8
 In-Call State Re-INVITE Handling 1-16
 Multi-Leg Call Control Finite State Machine 1-11
 Threads 1-6
 UAC Finite State Machine 1-14
 UAS Finite State Machine 1-12
In-Call State Re-INVITE Handling 1-16

Index (Continued)

INFO 1-4
INVITE 1-4
 event 1-7

M

Mascarpone 1-3
media handling 1-2
 concurrent media sessions 1-2
media server 1-4
Memory and Performance Impact 1-17
MLCC
 see Multi-Leg Call Control
MLCD
 see Multi-Leg Call Data
Multi-Leg Call Control
 state machine 1-7
 unused time 1-4
 worker thread 1-7
Multi-Leg Call Control Finite State Machine 1-11
Multi-Leg Call Data 1-7
MultiLegCallData 1-9

O

Open Settlement Protocol 1-2
OSP
 see Open Settlement Protocol
Overview 1-4

P

Problem Definition 1-2
Provisioning Server
 address and port numbers 1-5
PSLib
 b2bua.xml 1-5
Purpose 1-2

R

RADIUS
 see Remote Authentication Dial In User Service
RadiusPayload 1-9
Real Time Streaming Protocol 1-4
Recv 200 (BYE) 1-13
Recv ACK 1-13
Recv CANCEL 1-13
Recv INFO 1-13
Recv INVITE 1-12
Reference Documents 1-25
remaining prepaid time 1-4
Remote Authentication Dial In User Service 1-2
 AAA Transceiver Thread 1-7
 Access-Request message 1-4

accounting 1-7
Accounting Interim message 1-4
Accounting Start message 1-4
Accounting Stop message 1-4
 sending messages 1-7

replication 1-11
reusable design 1-3
RTSP
 see Real Time Streaming Protocol

S

Send >200 1-13
Send 18x 1-13
Send 200 (INVITE) 1-13
Send BYE 1-13
Session Initiation Protocol
 bis draft URL 1-2
 events 1-7
 stack 1-3
 thread 1-6
 Worker Thread FIFO queue 1-7
Session Intitiation Protocol
 RFC 1-2
SIP
 see Session Initiation Protocol
SipCallLegData 1-10
SipTransactionPeers 1-10
Source code 1-5
stand alone
 defined 1-2

T

Testing Considerations 1-23
Threads 1-6
timer expiry 1-4
Transceiver 1-6
transitions
 Recv 200 (BYE) 1-13
 Recv ACK 1-13
 Recv CANCEL 1-13
 Recv INFO 1-13
 Recv INVITE 1-12
 Send >200 1-13
 Send 18x 1-13
 Send 200 (INVITE) 1-13
 Send BYE 1-13

U

UaBase 1-10
UAC
 see User Agent Client
UaClient 1-10
UAS
 see User Agent Server

Index (Continued)

- UaServer 1-10
- uaState 1-10
- User account information 1-2
- user account information 1-2
- User Agent Client
 - state machine 1-7
 - within B2BUA 1-2
 - worker thread 1-7
- User Agent Client Finite State Machine 1-14
- User Agent Server
 - state machine 1-7
 - within B2BUA 1-2
 - worker thread 1-7
- User Agent Server Finite State Machine 1-12

V

- Vendor-Specific Attributes 1-4
- VOCAL
 - see Vovida Open Communications Applications Library
- VOCAL RADIUS stack 1-3
- Vovida Open Communications Applications Library 1-2
- VSAs
 - see Vendor-Specific Attributes

W

- Worker Thread FIFO queue 1-7

Index (Continued)
