
Juphoon Protocol Framework

Juphoon Phone Framework (JPF)

Published: July 2008

Please visit our website at <http://www.juphoon.com>

Use Juphoon JPF to Construct a SIP Phone

Juphoon System Software Co., Ltd.
<http://www.juphoon.com>
Tel: +86-574-87287820
Fax: +86-574-87304379
Document Number: 100-000-02-01

Copyright © 2008, Juphoon System Software Corporation.
All rights reserved.

Contents

USE JUPHOON JPF TO CONSTRUCT A SIP PHONE.....	1
1. INTRODUCTION.....	4
1.1 PURPOSES.....	4
1.2 FUNCTIONS.....	4
1.3 APPLICATIONS.....	4
2. THE MULTI-SERVICE FRAMEWORK.....	5
2.1 INTRODUCTION ON THE FRAMEWORK.....	5
2.2 COLLABORATIVE COMPONENTS.....	5
2.2.1 Components.....	6
2.2.2 Collaboration of components.....	7
2.3 USER COLLABORATION.....	7
2.4 CONCEPTS.....	8
2.4.1 Endpoint.....	8
2.4.2 Connection.....	9
2.4.3 Stream.....	9
2.4.4 Call.....	9
2.4.5 Call Flow.....	9
2.4.6 Session.....	10
2.4.7 Dialog.....	11
2.4.8 Transaction.....	11
2.5 THREAD MODEL.....	11
2.5.1 User Thread.....	12
2.5.2 JPF Thread.....	12
2.5.3 SUA Thread.....	14
2.5.4 Audio Thread.....	14
2.5.5 SIP Stack Thread.....	14
3. JPF COMPONENTS.....	15
3.1 OPERATION INTERFACES.....	15
3.2 STATE REPORTING.....	16
3.3 SERVICE PROCESSING.....	21
4. JPF SERVICE INTERFACES.....	22
4.1 ENDPOINT OPERATION INTERFACES.....	22
4.2 CONNECTION OPERATION INTERFACES.....	22
4.3 STATE NOTIFICATION.....	23
4.4 DATA MANAGEMENT.....	24
5. JPF FEATURES.....	25
5.1 REGISTRATION.....	25

5.2	BASIC CALL	27
5.3	CALLER ID	28
5.4	CALL HOLD.....	29
5.5	CONSULTATION HOLD	30
5.6	UNATTENDED TRANSFER.....	32
5.7	ATTENDED TRANSFER.....	34
5.8	CALL WAITING.....	36
5.9	MULTIPLE LINE APPEARANCE	37
5.10	MUTE	38
5.11	REDIAL.....	38
5.12	DO NOT DISTURB.....	39
6.	JPF SERVICES	41
7.	REGISTER	41
7.1	UN-REGISTER.....	41
7.2	OUTBOUND PROXY	41
7.3	PROXY/WWW AUTHENTICATION.....	42
7.4	STUN.....	42
7.5	DTMF	42

Tables

Table 2-1 Collaboration of SIP Phone Components	7
Table 4-1 Endpoint Operation Interfaces	22
Table 4-2 Connection Operation Interfaces	23
Table 4-3 Data Management Interfaces	24
Table 5-1 User Environments of Registration.....	27
Table 5-2 User Environments of Basic Call.....	28
Table 5-3 User Environment of Caller ID.....	29
Table 5-4 User Environments of Call Hold	30
Table 5-5 User Environments of Consultation Hold.....	32
Table 5-6 User Environments of Unattended Transfer	34
Table 5-7 User Environments of Attended Transfer	36
Table 5-8 User Environments of Call Waiting.....	37
Table 5-9 User Environments of Multiple Line Appearance	38
Table 5-10 User Environments of Mute.....	38
Table 5-11 User Environments of Redial.....	39
Table 5-12 User Environments of Do Not Disturb	40
Table 6-1 Interfaces of handling Register related data.....	41
Table 6-2 Proxy Data Operation Interfaces.....	42

Figures

Figure 2-1 Modules of SIP Phone Multi-Service Framework	5
Figure 2-2 Collaborative Components of SIP Phone Multi-Service Framework.....	6
Figure 2-3 User Interaction in SIP Phone Multi-Service Framework	8
Figure 2-4 Basic Concepts in SIP Phone Multi-Media Framework.....	8
Figure 2-5 Basic Call Flows.....	10
Figure 2-6 The Thread Model of the Multi-Service Framework	11
Figure 3-1 JPF Architecture	15
Figure 5-1 Order of Registration.....	26
Figure 5-2 Order of Basic Call.....	27
Figure 5-3 Order of Caller ID	28
Figure 5-4 Order of Call Hold.....	29
Figure 5-5 Order of Consultation Hold	31
Figure 5-6 Order of Unattended Transfer	33
Figure 5-7 Order of Attended Transfer	35
Figure 5-8 Order of Call Waiting.....	36
Figure 5-9 Order of Multiple Line Appearance	37
Figure 5-10 Order of Mute.....	38
Figure 5-11 Order of Do Not Disturb	39

1. Introduction

1.1 Purposes

SIP Phone Multi-Service Framework was created for the following purposes:

- Providing uniform application framework for various SIP Phone clients. One set of code can supply services to many clients like Softphone, SIP phone, and etc.
- Separating the service control from the signaling control.
- Separating service implementation from the practical media control

1.2 Functions

The SIP Phone Multi-Service Framework has the following functions:

- Audio calls
- Video calls
- Interoperation between different users
- Standard SIP features
- Standard basic SIP call features including Register, Reliable Response, Heart Beat, and etc.
- Value-added services including Call Hold, Call Transfer, 3-way Conference, and etc.
- Other SIP services including IM, Presence, PTT, and etc.

1.3 Applications

SIP Phone Multi-Service Framework can be applied to the following devices:

- SIP Audio/Video Telephone
- SIP Software Phone
- Residential Gateway

2. The multi-service framework

2.1 Introduction on the framework

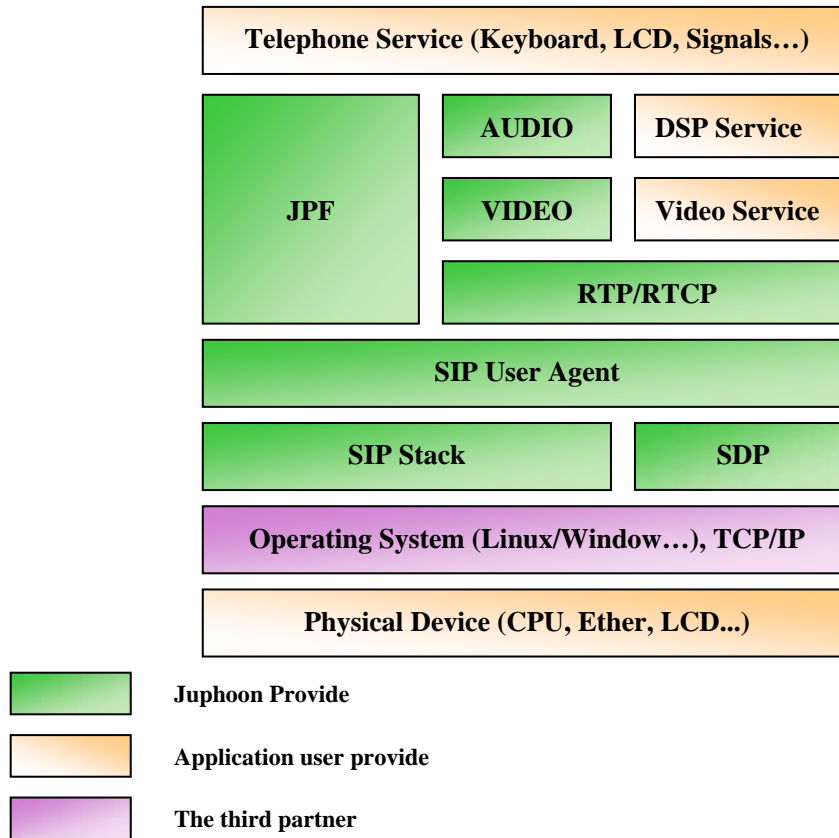


Figure 2-1 Modules of SIP Phone Multi-Service Framework

2.2 Collaborative components

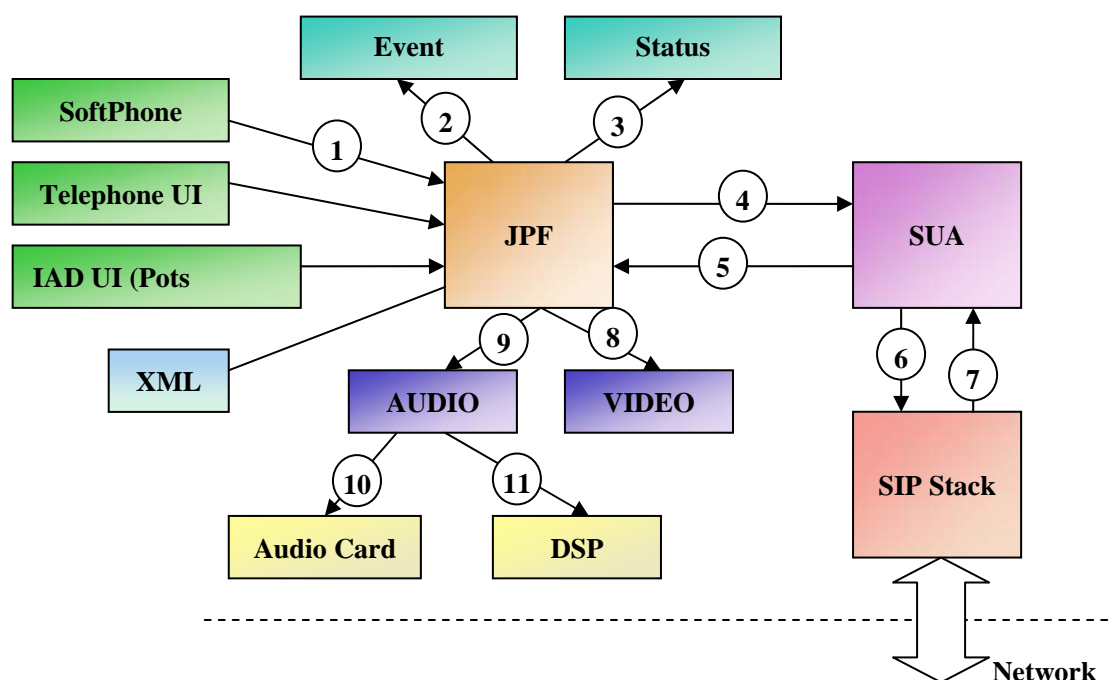


Figure 2-2 Collaborative Components of SIP Phone Multi-Service Framework

2.2.1 Components

The SIP Phone mainly consists of the following components

- SIP stack, responsible for the event control of SIP signaling and the session control.
- SUA (SIP User Agent), responsible for the service signaling control of more than one SIP call (a call between two SIP phones).
- JPF (Juphoon Phone Framework), responsible for the service management and media control like opening or closing media channels and the media negotiation in the SIP signaling of each endpoint, reporting incoming calls and related operation states to the user.
- Softphone GUI, Telephone UI, IAD UI are nearly user-level applications. For instance, Softphone is an implementation of phone interface. It has a call button, a button for answering calls, and etc. Telephone UI has buttons like pick up, hang up, dial, switch hook flash and etc.
- AUDIO and VIDEO is an abstract layer of media control. By using interfaces (like Audio_StrmCreate used to create a media stream) provided by AUDIO and VIDEO, JPF can control media channels (open or close) and the media streaming directions (send only or receive only). AUDIO is responsible for the implementation of driver interfaces of real audio cards and DSP. It also maintains the RTP media control. And all Audio QoS related functions are realized in this module.

2.2.2 Collaboration of components

NO	Sender	Receiver	Discription
1	Softphone GUI	JPF	JPF provides call related interfaces for users and the Softphone GUI can then implement services according to the user's operations.
2, 3	JPF	Softphone GUI	JPF provides incoming call and service related callbacks and callbacks which indicate service states. Softphoen GUI only needs to implement related user operations in the callbacks, like displaying an answering dialog box for the user to answer an incoming call on the indication of the call.
4	JPF	SUA	SUA provides message interfaces of call out services.
5	SUA	JPF	SUA provides interfaces of service control messages from the peer.
6	SUA	SIP Stack	SUA uses SIP Stack primitive interfaces to create and manage sessions.
7	SIP Stack	SUA	SIP Stack provides interfaces of session control messages from the peer.
8, 9	JPF	AUDIO, VIDEO	JPF uses interfaces of the media layer to control media channels.
10	AUDIO	Audio cards	AUDIO uses the audio card driver to play sound and receive data samples.
11	AUDIO	DSP	AUDIO uses DSP driver to play sound and receive data samples.

Table 2-1 Collaboration of SIP Phone Components

2.3 User collaboration

The user needs to provide the following collaborative components:

- **DSP Service**
provides audio Codec like G.711 A/U, G.723.1, G.729 and etc. and audio QoS components like echo cancellation, mute suppression, and etc.
- **Video Service**
provides audio sampling and display components.
- **Telephone Service**
Provides driver interfaces including pick up, hang up, and dial and LCD components.

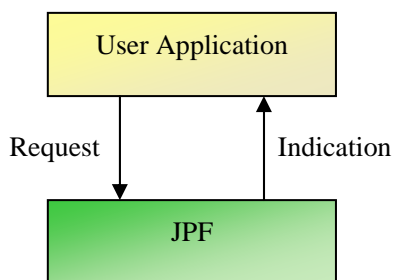


Figure 2-3 User Interaction in SIP Phone Multi-Service Framework

2.4 Concepts

The basic concepts in the framework are shown in figure 2-4 below:

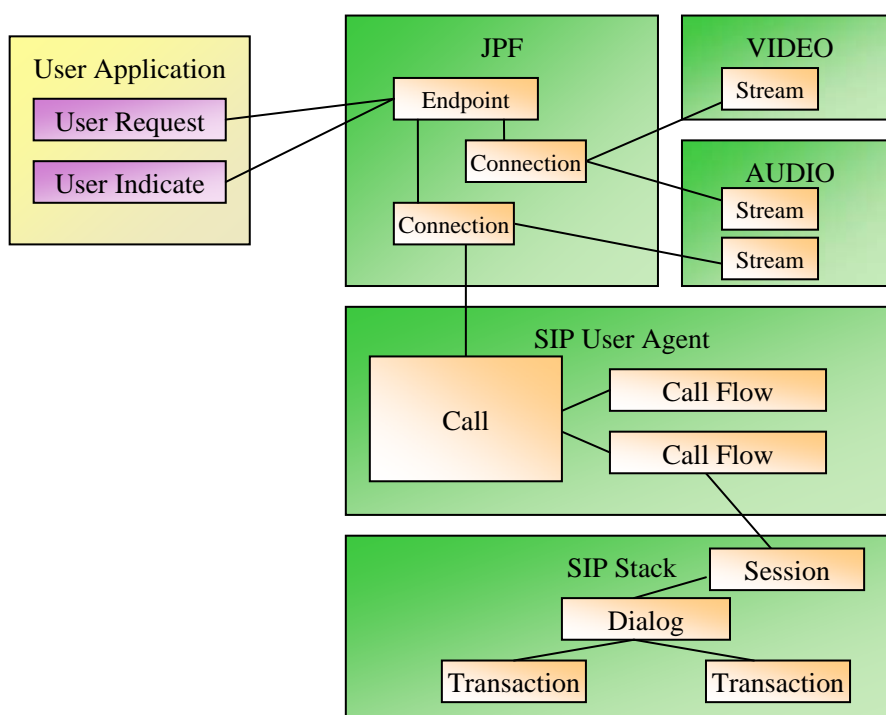


Figure 2-4 Basic Concepts in SIP Phone Multi-Media Framework

2.4.1 Endpoint

An endpoint is a logic user. It can be a login user, a SIP phone, or a port of a SIP gateway. Each endpoint has some common device properties like local IP, local SIP signaling port, device NAT, STUN config, and etc. Each endpoint also has its special properties like user URI, registered account and password. Each endpoint can implement different services like basic calls, conference call service, Instant Messaging, and etc. The user can use the endpoint interfaces to register the endpoint or retrieve the endpoint information. Practical services are related to practical connections.

Each endpoint has a unique ID number. The user gets the endpoint ID when creating it. Operations are carried out on the endpoint identified by a certain endpoint ID. So the user has to set up a correlation control block to store endpoint IDs.

2.4.2 Connection

The connection between two endpoints indicates their service relationship, such as basic call, Call Transfer, IM and others. A connection only provides services that do not conflict with other services, such as putting a basic call on hold or transferring a call. An instant message can be sent in a connection or different connections.

An endpoint keeps more than one connection if it is carrying out a service involving more than two endpoints like 3-way conference where the conference service activates audio mix for the audio stream. In this case, each connection the endpoint keeps represents a basic call with a remote endpoint.

Each connection keeps some state information about the two endpoints involved. The user can use the connection interfaces to get the addresses of the two endpoints and the call states of the connection. The connection also provides service interfaces, such as Call Hold, Call Transfer, and etc.

Each connection has a unique ID number in the device. The user should get the connection ID when creating the connection. Connection operations are carried out on the connection identified by a certain connection ID. So the user has to set up a correlation control block to store connection IDs.

The user can provide control handles to build correlations with newly created connections. When receiving an event notification the user can use the connection interfaces to get the event related handle to determine the location of the user management structure.

2.4.3 Stream

Stream means media data stream, including audio data and video data. Each stream has directions. There are four main directions: receive-only, send-only, receive and send, none. The user can use

2.4.4 Call

Call is an informal term, referring to communication between endpoints and often used to establish multi-media sessions. Semantically *call* belongs to protocols. The user doesn't need to understand calls. What users need to care are services which are completed by protocol calls. On the module level, call control exists in the SIP User Agent (SUA) module.

A connection between two endpoints can implement many services and it maps into a call in the SUA. Service control related messages are maintained and transmitted by calls in the SUA.

In SUA, every call has a unique ID number. A call control block of the communication between JPF and SUA is identified by a connection ID and a call ID.

2.4.5 Call Flow

Call Flow indicates that a call is a process. Basic call flows are shown in the figure below:

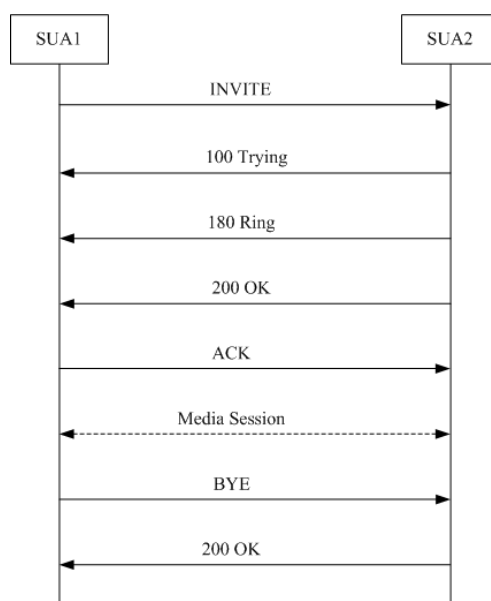


Figure 2-5 Basic Call Flows

Call flows during the realization of SUA can be categorized by call function into basic call flow, call transfer flow, and etc.

In a call there can be more than one call flows that don't conflict with each other. Call flows like registration flow and basic call flow that conflict with each other belong to different calls. Basic call flows can coexist with call transfer flows. There may be two flows in a call when the call is being transferred.

The flow designs may be a little different from the practical services. For example the basic session service and call hold service are in the same basic call flow.

In SUA, every flow has a unique ID number. JPF doesn't need to care the flow IDs. It only needs to know the call IDs.

2.4.6 Session

Session is a basic concept in SIP. A multimedia session is a set of multimedia senders and receivers and the data streams flowing from senders to receivers. A multimedia conference is an example of a multimedia session.

In the multi-service framework, a session can contain more than one Dialog. In SIP protocol, a session is a set of Dialogs with the same Call-Ids and From-Tags and different To-Tags. On the module level, the call control exists in the SIP Stack module.

Every session has its unique ID number and its user ID – the calling ID in SUA.

When SUA is originating a new call as a caller, it needs to pass the call ID to the SIP Stack. The SIP Stack passes the call ID to SUA when responding to the call. Thus SUA and SIP Stack has built a correlation between the call and the session. When SUA acts as a callee, the whole process of building a correlation between SUA and SIP is in reverse order.

2.4.7 Dialog

Dialog is a basic concept in the SIP Stack. It is a peer-to-peer SIP relationship between two users that persists for some time. A dialog is identified by a From-Tag and a To-Tag in the SIP protocol. A tag is a random encrypted sequence and should be globally unique.

A dialog is established by SIP messages, such as INVITE, SUBSCRIBE, and REFER. Please refer to RFC3261, RFC3265, and RFC3515 of the dialog establishment.

Each dialog has a unique ID number and its user ID – the call ID in SUA.

When SUA is originating a new call as a caller, it needs to pass the call ID to the SIP Stack. The SIP Stack passes the call ID to SUA when responding to the call. Thus SUA and SIP Stack has built a correlation between the call and the session. When SUA acts as a callee, the whole process of building a correlation between SUA and SIP is in reverse order.

2.4.8 Transaction

Transaction is a basic concept in the SIP Stack. It consists of a request and all the responses to the request. In SIP stack if the final response to an INVITE request indicates a failure, then the INVITE transaction will include an ACK request for the failure.

Each transaction has a unique ID number and it doesn't need to keep the user ID. But the user has to keep the transaction ID when receiving a transaction notification and pass the transaction ID to the SIP Stack when sending a response to the request. If the user only wants to send a request, then setting the transaction ID to ZMAXULONG and pass the ID to the SIP Stack.

2.5 Thread model

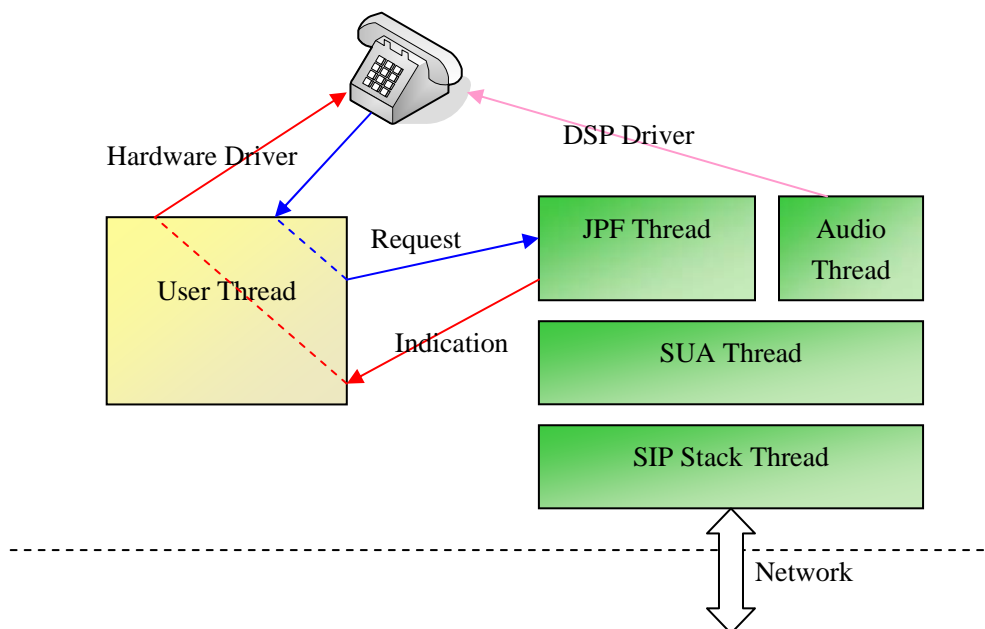


Figure 2-6 The Thread Model of the Multi-Service Framework

2.5.1 User Thread

The user thread handles driving signals of the transmitting and processing of events between the JPF thread and the user interface. When the user is carrying out an operation, the user thread will receive many kinds of driving signals and interface events. It then informs the JPF thread of these events. For example, when the user thread receives a notification of the state of the JPF thread, it will originate a driver signal and inform the user of the JPF state through the interface application. For example, if a SIP phone acting as the user, the user thread processes or originates the keyboard driving signal, LCD driving signal and other related signals and passes calling state events of the JPF thread.

2.5.2 JPF Thread

The user application directly invokes the interfaces provided by JPF to initiate the JPF task. Please see the following example about how to directly invoking JPF initiating functions.

```
ZINT main()
{
    /* init zos config */
    Zos_SysCfgInit();

    /* zos system init */
    if (Zos_SysInit() != ZOK)
        return -1;

    /* jpf database init */
    Jpf_DbInit(JCPE_CFG_FILE_NAME);

    /* sdp abnf init */
    if (Sdp_AbnfInit() != ZOK)
        return -1;

    /* start utal */
    if (Utal_Start() != ZOK)
        return -1;

    /* start sip stack */
    if (Sip_Start() != ZOK)
        return -1;
    /* start rtp stack */
    if (Rtp_Start() != ZOK)
        return -1;

    /* start sua */
    if (Sua_Start() != ZOK)
        return -1;

    /* start audio */
    if (Audio_Start() != ZOK)
        return -1;

    /* start dns */
    if (Dns_Start() != ZOK)
        return -1;

    /* start stun */
    if (Stun_Start() != ZOK)
        return -1;

    /* start jpf */
}
```

```
if (Jpf_Start() != ZOK)
    return -1;

/* start jcphone */
if (Jcpe_Start() != ZOK)
    return -1;

/* init zos shell */
if (Zsh_Init(ZNULL) != ZOK)
    return -1;

return 0;
}
```

2.5.3 SUA Thread

The SUA thread is the SIP User Agent, processing the SIP call signaling flows including basic call flows, registration flows, and REFER flows. For the JPF thread, the SUA thread is a layer entity which hides the call details of flow processing related to the call protocol.

When processing an event from the JPF thread SUA will, according to the call flow state, send a corresponding SIP Stack primitive event to the SIP Stack. When processing primitive event from the SIP Stack SUA will, according to the call flow state, send a corresponding event to the JPF thread.

2.5.4 Audio Thread

The audio thread is an abstract audio processing layer, hiding details on audio codec and quality processing from the JPF user.

The audio thread processes the audio equipment and codec on the basis of a specific SIP endpoint. For instance, if a SIP phone uses a DSP chip, then the user needs to modify the responding driver interfaces of playing sound in the Audio layer and modify the interfaces for processing sampled audio data.

The JPF user only needs to use the audio module interfaces to implement the audio functions like playing the ringing tone by invoking the **Audio_TonePlay** interface without sending an event to the audio thread.

2.5.5 SIP Stack Thread

The SIP Stack thread processes SIP sessions, dialogs, transactions, and transmission processes. SIP Stack finds the session and dialog that a SIP message or inner event (like waiting-timeout) belongs to. A SIP session and dialog in the SIP Stack keep their user ID to identify their service object.

3. JPF components

Juphoon Phone Framework (JPF) is a SIP Phone applicable framework implemented on the basis of the SIP Phone Multi-Service Framework. It mainly consists of the following components:

- **Endpoint / Connection API**
Carries out user operations on the inner objects and provides interfaces for obtaining states of the inner objects.
- **Status Report**
Reports state changes of the inner objects to the user.
- **Application Process**
Ensures the proper process of the multiple services between JPF, the user, and SUA.

The architecture of JPF is shown below:

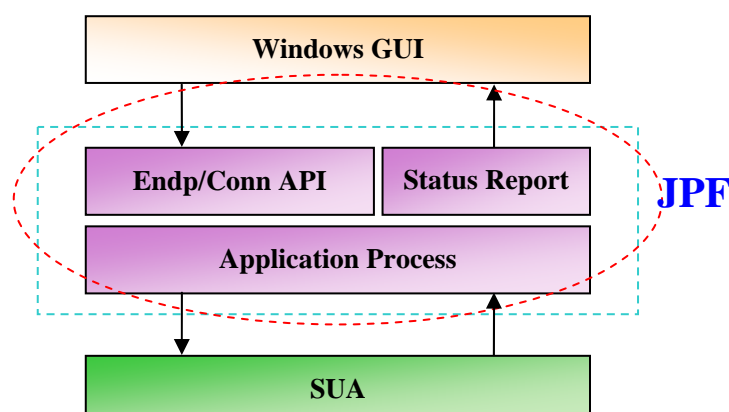


Figure 3-1 JPF Architecture

3.1 Operation interfaces

Operation interfaces including order interfaces and state interfaces. Order interfaces provide services like initiating a call, answering a call. State interfaces allow the user to get states of the operational objects, such as call states, SIP address information of the peer, and etc.

The two types of interfaces have different ways of usage. The order interfaces are invoked by the user when driving signals are received or interface events are triggered. The state interfaces are invoked by the user when the user receives a state report to get related information like the SIP peer URI and then the user can display related information and decides whether to carry out the next order.

The operation interfaces can be divided into endpoint interfaces and connection interfaces on the basis of operational objects. Though these two groups of interfaces provide different services, they

have similar ways to operate. The operation order is that creating an endpoint at first by using the endpoint interfaces and then carrying out the connection operations.

3.2 State reporting

States reporting informs the user of operation results and state changes of the inner operational objects by callback functions. The user decides what state message it is by the state type, gets related information by invoking the state interfaces and use the order interfaces to carry out the next order.

State reports can be divided into registration state report and connection state report on the basis of the operational objects. Usually, the user has to register two callback functions for registration state report and connection state report to get the state report information. The registration interface is **Jpf_UsrRegCallback**. The user invokes this interfaces to register the user-implemented interface for processing state information with JPF. Please read the following example:

```
/* jcphone framework event process register response */
ZINT Jcpe_ActFeProcRegStat(ZULONG dwEndpId, ZUCHAR ucStatType)
{
    if (ucStatType == EN_JPFE_REG_STAT_NOT_REGED)
    {
        JPF_LOG_INFO((JPF_LOGID, "endpoint not registered.));
    }
    else if (ucStatType == EN_JPFE_REG_STAT_REG_ACPTED)
    {
        JPF_LOG_INFO((JPF_LOGID, "endpoint register accepted.));
    }
    else if (ucStatType == EN_JPFE_REG_STAT_REG_REJED)
    {
        JPF_LOG_INFO((JPF_LOGID, "endpoint register rejected.));
    }
    else if (ucStatType == EN_JPFE_REG_STAT_REG_ERR)
    {
        JPF_LOG_INFO((JPF_LOGID, "endpoint register error.));
    }
    else if (ucStatType == EN_JPFE_REG_STAT_UNREG_ERR)
    {
        JPF_LOG_INFO((JPF_LOGID, "endpoint unregister error.));
    }

    return ZOK;
}
```

```
/* jcphone framework event process connection status */
ZINT Jcpe_ActFeProcConnStat(ZULONG dwConnId, ZUCHAR ucStatType)
{
    switch (ucStatType)
    {
        case EN_JPFE_CONN_STAT_CALLOUT:
            JPF_LOG_INFO((JPF_LOGID, "connection <%p> is callout.",
dwConnId));
            break;
        case EN_JPFE_CONN_STAT_ALERTED:
            /* play ringback tone */
            Jpf_ConnPlayTone(EN_AUDIO_TONE_RING_BACK,
JPF_TONE_FOREVER);

            JPF_LOG_INFO((JPF_LOGID, "connection <%p> is alerted.",
dwConnId));
            break;
    }
}
```

```
case EN_JPFE_CONN_STAT_CALLIN:
    /* play ring tone */
    Jpf_ConnPlayTone(EN_AUDIO_TONE_RING, JPF_TONE_FOREVER);
    Zos_Printf("%s\r\n",
              "Calling Even! Press 'a' to accept, or 'r' to
reject");

    /* save connection id */
    g_dwJcpeConnId = dwConnId;

    JPF_LOG_INFO((JPF_LOGID, "connection <%p> remote is call
in.", dwConnId));
    break;
case EN_JPFE_CONN_STAT_CONNED:
    /* stop tone */
    Jpf_ConnStopTone();

    /* save connection id */
    g_dwJcpeConnId = dwConnId;

    JPF_LOG_INFO((JPF_LOGID, "connection <%p> is connected.",
dwConnId));
    break;
case EN_JPFE_CONN_STAT_TERMINATE:
    /* stop tone */
    Jpf_ConnStopTone();

    /* delete the connection id */
    g_dwJcpeConnId = ZMAXULONG;

    JPF_LOG_INFO((JPF_LOGID, "connection <%p> is terminated.",
dwConnId));
    break;
case EN_JPFE_CONN_STAT_MDFYING_ACPTED:
case EN_JPFE_CONN_STAT_MDFYING_ERR:
case EN_JPFE_CONN_STAT_MDFYED:
case EN_JPFE_CONN_STAT_HOLD_ERR:
case EN_JPFE_CONN_STAT_CHOLD_TERMED:
case EN_JPFE_CONN_STAT_CHOLD_ERR:
case EN_JPFE_CONN_STAT_REFERED:
case EN_JPFE_CONN_STAT_REFER_TERMED:
case EN_JPFE_CONN_STAT_REFER_ERR:
case EN_JPFE_CONN_STAT_TRSF_ACPTED:
case EN_JPFE_CONN_STAT_TRSF_TERMED:
```

```
case EN_JPFE_CONN_STAT_TRSF_ERR:
case EN_JPFE_CONN_STAT_HEARTBEAT_ERR:
case EN_JPFE_CONN_STAT_REDIRECTED:
    break;
default:
    JPF_LOG_ERROR((JPF_LOGID, "unknown status type."));
    break;
}

return ZOK;
}
```

In the example above, Jcpe_ActFeProcRegStat is the callback function implemented by the user for getting registration state information and Jcpe_ActFeProcConnStat is the callback function implemented by the user for getting connection state information. The user has to register these two callback functions with JPF. The following example shows how to register these two callback functions:

```
/* jphone framework process register response */
typedef ZINT (*PFN_JPFPROCREGSTAT)(ZULONG dwEndpId, ZUCHAR
ucStatType);

/* jphone framework process connection status */
typedef ZINT (*PFN_JPFPROCCONNSTAT)(ZULONG dwConnId, ZUCHAR
ucStatType);

/* jphone framework event process by user */
typedef struct tagJPFE_PROC
{
    PFN_JPFPROCREGSTAT pfnRegStat; /* register response process */
    PFN_JPFPROCCONNSTAT pfnConnStat; /* connection status process */
} ST_JPFE_PROC;

ZINT Jcpe_RegCallback(PFN_JPFPROCREGSTAT pfnRegStat,
                      PFN_JPFPROCCONNSTAT pfnConnStat)
{
    ST_JPFE_PROC stEvtProc;

    /* register call back */
    stEvtProc.pfnRegStat = pfnRegStat;
    stEvtProc.pfnConnStat = pfnConnStat;

    return Jpf_UsrRegCallback(&stEvtProc);
}

ZINT main()
{
    char cmd[128];
    ST_ZOS_SSTR stCalleeUri;

    if (Jcpe_Initialize() != ZOK)
        return -1;

    if (Jcpe_RegCallback(Jcpe_ActFeProcRegStat,
                        Jcpe_ActFeProcConnStat) != ZOK)
        return -1;

    if (Jcpe_Start() != ZOK)
        return -1;

    if (Jcpe_EndpCreate(&g_dwJcpeEndpId) != ZOK)
        return -1;
}
```

```
.....  
}
```

In the example above, Jcpe_RegCallback registers Jcpe_ActFeProcRegStat and Jcpe_ActFeProcConnStat with JPF by invoking the registration interface **Jpf_UsrRegCallback**.

3.3 Service processing

The finite state machines inside JPF process the following types of services:

- Register
The user registration service
- Basic Call
The basic call service
- Consultation Hold
The Consultation Hold service
- Attended Transfer
The Attended Transfer service
- Unattended Transfer
The Unattended Transfer service
- Refer
The Refer service – receiving a REFER request from the peer

These services coexist without any conflict – the basic call service co-exists with other value-added service like Call Hold.

New JPF services can be added and maintained in the service processing modules. When JPF receives a user event or SUA event, it will activate the corresponding FSM, and then will send a notification according to the service state to the user or SUA. In fact, the user state event callback function is implicitly invoked when the service is being processed.

4. JPF service interfaces

4.1 Endpoint operation interfaces

Main endpoint operation interfaces are shown in Table 4-1 below:

Interface	Description
Jpf_EndpCreate	Creates an endpoint
Jpf_EndpDelete	Deletes an endpoint
Jpf_EndpReg	Registers an endpoint with the registrar
Jpf_EndpUnreg	Deregisters an endpoint with the registrar
Jpf_EndpGetCookie	Gets the user handle relating to a specific endpoint
Jpf_EndpGetLocalAddr	Gets the SIP address of an endpoint
Jpf_EndpGetLocalUri	Gets the SIP URI of an endpoint
Jpf_EndpGetRegState	Gets the registration state of an endpoint
Jpf_EndpGetConnNum	Gets the number of valid connections in an endpoint
Jpf_EndpGetConnItem	Gets the ID of a connection by the connection's index

Table 4-1 Endpoint Operation Interfaces

4.2 Connection operation interfaces

Main connection operation interfaces are shown in Table 4-2 below:

Interface	Description
Jpf_ConnCall	Originates a call
Jpf_ConnAnswer	Answers a call
Jpf_ConnTerminate	Terminates a call
Jpf_ConnHold	Puts a call on hold
Jpf_ConnCHold	Puts a call on Consultation Hold
Jpf_ConnUnhold	Unholds a call
Jpf_ConnUTrsf	Unattended transfers a call
Jpf_ConnATrsf	Attended transfers a call
Jpf_ConnReferAcpt	Accepts a transferred call
Jpf_ConnReferRej	Refuses a transferred call
Jpf_ConnPlayTone	Starts playing the audio file.
Jpf_ConnStopTone	Stops playing the audio file
Jpf_ConnStartDtmf	Starts sending DTMF signals
Jpf_ConnStopDtmf	Stops sending DTMF signals
Jpf_ConnGetMuted	Gets the audio Mute state
Jpf_ConnSetMuted	Sets the audio Mute state
Jpf_ConnGetHolded	Gets the call-hold state

Jpf_ConnGetEndpld	Gets ID of the local endpoint which a specific connection involves
Jpf_ConnGetCookie	Gets the user handle relating to a specific connection
Jpf_ConnGetType	Gets the type of a connection
Jpf_ConnGetState	Gets the state of a connection
Jpf_ConnGetCHoldState	Gets the Consultation Hold state of a connection
Jpf_ConnGetUTrsfState	Gets the Unattended Transfer state of a connection
Jpf_ConnGetATrsfState	Gets the Attended Transfer state of a connection
Jpf_ConnGetReferState	Gets the Transfer state of a connection
Jpf_ConnGetAudioState	Gets the state of the audio media stream in a connection
Jpf_ConnGetVideoState	Gets the state of the video media stream in a connection
Jpf_ConnGetLocalAddr	Gets the local SIP address
Jpf_ConnGetPeerAddr	Gets the peer SIP address
Jpf_ConnGetLocalUri	Gets the local SIP URI
Jpf_ConnGetPeerUri	Gets the peer SIP URI
Jpf_ConnGetCHoldUri	Gets the Consultation Hold URI
Jpf_ConnGetReferToUri	Gets the Refer To URI
Jpf_ConnGetReferByUri	Gets the Refer By URI

Table 4-2 Connection Operation Interfaces

4.3 State notification

State notification needs the following two callback functions to be registered.

- **PFN_JPFPROCREGSTAT**
- **PFN_JPFPROCCONNSTAT**

One of these two functions is responsible for receiving registration states, the other connection states. Details of these two functions are shown below:

```

/* jphone framework process register response */
typedef ZINT (*PFN_JPFPROCREGSTAT)(ZULONG dwEndpId, ZUCHAR
ucStatType);

/* jphone framework process connection status */
typedef ZINT (*PFN_JPFPROCCONNSTAT)(ZULONG dwConnId, ZUCHAR
ucStatType);

/* jphone framework event process by user */
typedef struct tagJPFE_PROC
{
    PFN_JPFPROCREGSTAT pfnRegStat; /* register response process */
    PFN_JPFPROCCONNSTAT pfnConnStat; /* connection status process */
} ST_JPFE_PROC;

```

The definition of the registered call function **Jpf_UsrRegCallback** is shown below:

```

/* jphone framework register callback */
ZINT Jpf_UsrRegCallback(ST_JPFE_PROC *pstEvtProc);

```

4.4 Data management

JPF provides several interfaces for data management. These interfaces need to be modified according to the customers' requirements.

Table 4-3 lists part of the interfaces for data management. For more information please refer to JPF Function Definition.

Interface	Description
Jpf_DbGetLocallp	Gets the local IP address
Jpf_DbGetLogLevel	Gets the current log level
Jpf_DbGetUsrName	Gets the local user name
Jpf_DbGetUsrUri	Gets the local user URI
Jpf_DbSetLocallp	Sets the local IP address
Jpf_DbSetLogLevel	Sets the current log level
Jpf_DbSetUsrName	Sets the local user name
Jpf_DbSetUsrUri	Sets the local user URI

Table 4-3 Data Management Interfaces

5. JPF features

JPF provides the following SIP features:

- Register
- Basic Call
- Caller ID
- Call Hold
- Consultation Hold
- Unattended Transfer
- Attended Transfer
- Call Waiting
- Multiple Line Appear
- Mute
- Redial
- Do Not Disturb

5.1 Registration

The registration service allows the user to register or deregister. The process of the user application and JPF and other modules working together to perform the registration service is shown below:

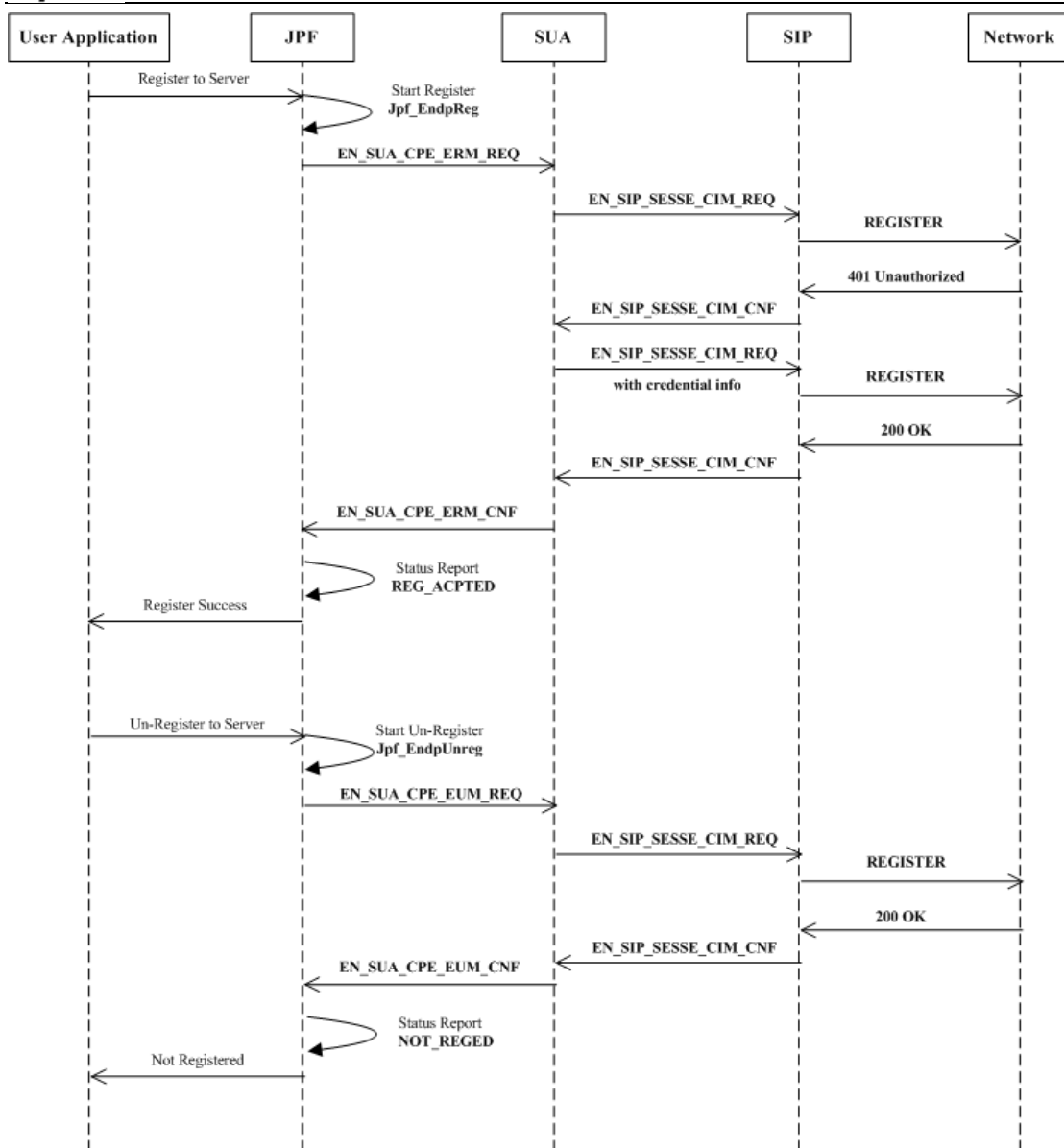


Figure 5-1 Order of Registration

Environment	Description
Prerequisite	A user endpoint must have been created by Jpf_EndpCreate .
Callback function	The user needs to implement the callback function for notifying registration states. The type of the callback function is PFN_JPFPROCREGSTAT . When being initialized, this callback function has to be registered by using Jpf_UsrRegCallback .
Data capture	Use the state type having been sent by the callback function to decide what the registration state is. The user can use Jpf_EndpGetCookie to get the handle of the user application and use Jpf_EndpGetRegState to get the registration state.
Supplement	The user can use Jpf_DbSetRegDomain , Jpf_DbSetRegAccount , Jpf_DbSetRegPasswd , Jpf_DbSetProxyIp , Jpf_DbSetProxyUdpPort ,

	<p>Jpf_DbSetProxyTcpPort to config the registration related parameters.</p> <p>The user can use interface Jpf_DbSetRegEnable to set the parameter indicating whether to register or not.</p>
--	--

Table 5-1 User Environments of Registration

5.2 Basic Call

Basic Call is the basic call function. The process of a basic call carried out by the user application, JPF, and other modules are shown in Figure 5-2 below:

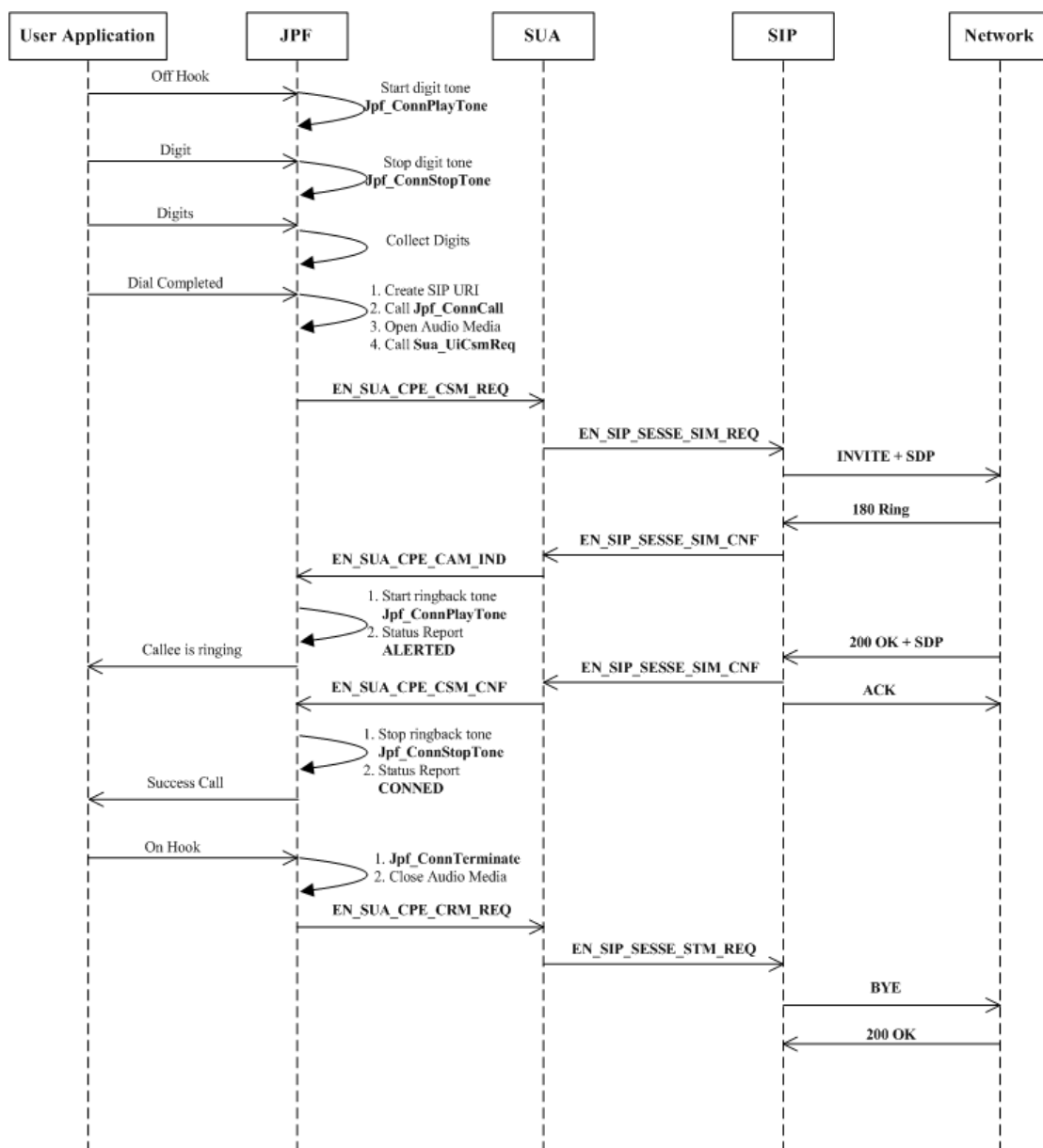


Figure 5-2 Order of Basic Call

Environment	Description
Prerequisite	A user endpoint must have been created by Jpf_EndpCreate .
Callback function	The user needs to implement the callback function for notifying connection states. The type of the callback function is PFN_JPFPROCCONNSTAT . When being initialized, this callback function has to be registered by using Jpf_UsrRegCallback .
Data capture	Use the state type having been sent by the callback function to decide what the state of a connection is. The user can use Jpf_ConnGetEndpId in the callback function to get the ID of the corresponding endpoint, Jpf_ConnGetCookie to get the user handle, Jpf_ConnGetState to get the connection state, and Jpf_ConnGetLocalAddr and Jpf_ConnGetPeerAddr to get the information of the two endpoints linked with each other by the connection.
Supplement	The user can use Jpf_DbSetUsrName and Jpf_DbSetUsrUri to set the user information and use Jpf_DbSetUsedCodecs , Jpf_DbSetAecEnable , Jpf_DbSetAnrEnable , Jpf_DbSetAgcEnable , Jpf_DbSetAsdEnable , and Jpf_DbSetLiveToneEnable to set the media capabilities and media options.

Table 5-2 User Environments of Basic Call

5.3 Caller ID

Caller ID is a piece of the caller information which can be displayed when the user receives a new call. The order of caller ID is shown below:

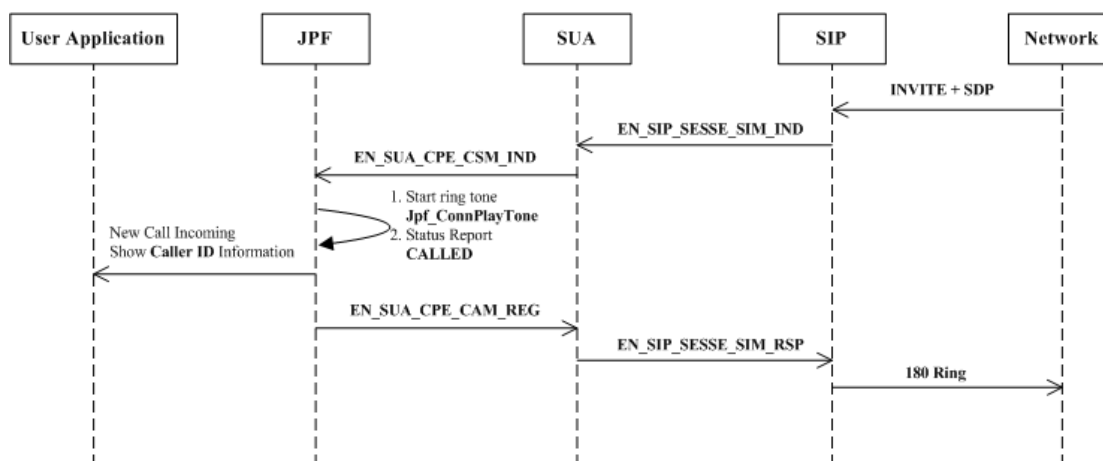


Figure 5-3 Order of Caller ID

Environment	Description
Prerequisite	A user endpoint must have been created by Jpf_EndpCreate .
Callback function	The user needs to implement the callback function for notifying connection states. The type of the callback function is PFN_JPFPROCCONNSTAT . When being initialized, this callback function has to be registered by using Jpf_UsrRegCallback .

Data capture	<p>Use the state type having been sent by the callback function to decide what the state of a connection is.</p> <p>The user can use Jpf_ConnGetEndpId in the callback function to get the ID of the corresponding endpoint, Jpf_ConnGetCookie to get the user handle, Jpf_ConnGetState to get the connection state, Jpf_ConnGetPeerAddr and Jpf_ConnGetPeerUri to get the caller information of the connection.</p>
Supplement	<p>The user can use Jpf_DbSetUsrName and Jpf_DbSetUsrUri to set the user information.</p>

Table 5-3 User Environment of Caller ID

The format of Caller ID is [UserName] <sip:UserInfo@UserAddress>. UserName is optional and the detailed information about Caller ID is contained in the From Header of INVITE message. For example:

"Bob Liu" <sip:bob@juphoon.com>
 <sip:+8657587304379@juphoon.com>

5.4 Call Hold

Call Hold means putting a call on hold. The process of Call Hold is shown below:

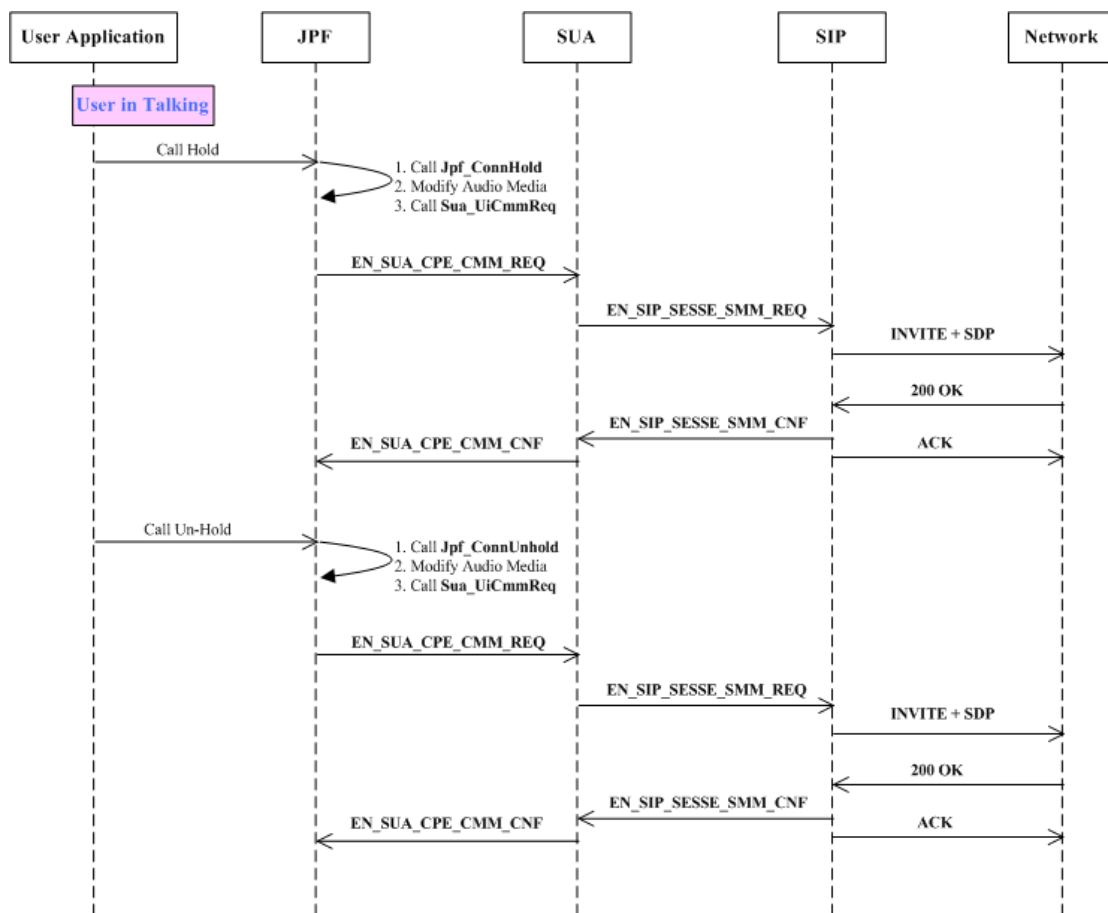


Figure 5-4 Order of Call Hold

Environment	Description
Prerequisite	A user endpoint must have been created by Jpf_EndpCreate . And the user connection must be in the normal talk state.
Callback function	The user needs to implement the callback function for notifying connection states. The type of the callback function is PFN_JPFPROCCONNSTAT . When being initialized, this callback function has to be registered by using Jpf_UsrRegCallback .
Data capture	Use the state having been sent by the callback function to decide the states of a connection mainly including whether an error has occurred when putting a call on hold and if the connection is terminated.
Supplement	None.

Table 5-4 User Environments of Call Hold

5.5 Consultation Hold

Consultation Hold allows the user to place a call on hold and use the phone to call another party to consult privately, and afterward, return to the original call. The process of Consultation Hold is shown below:

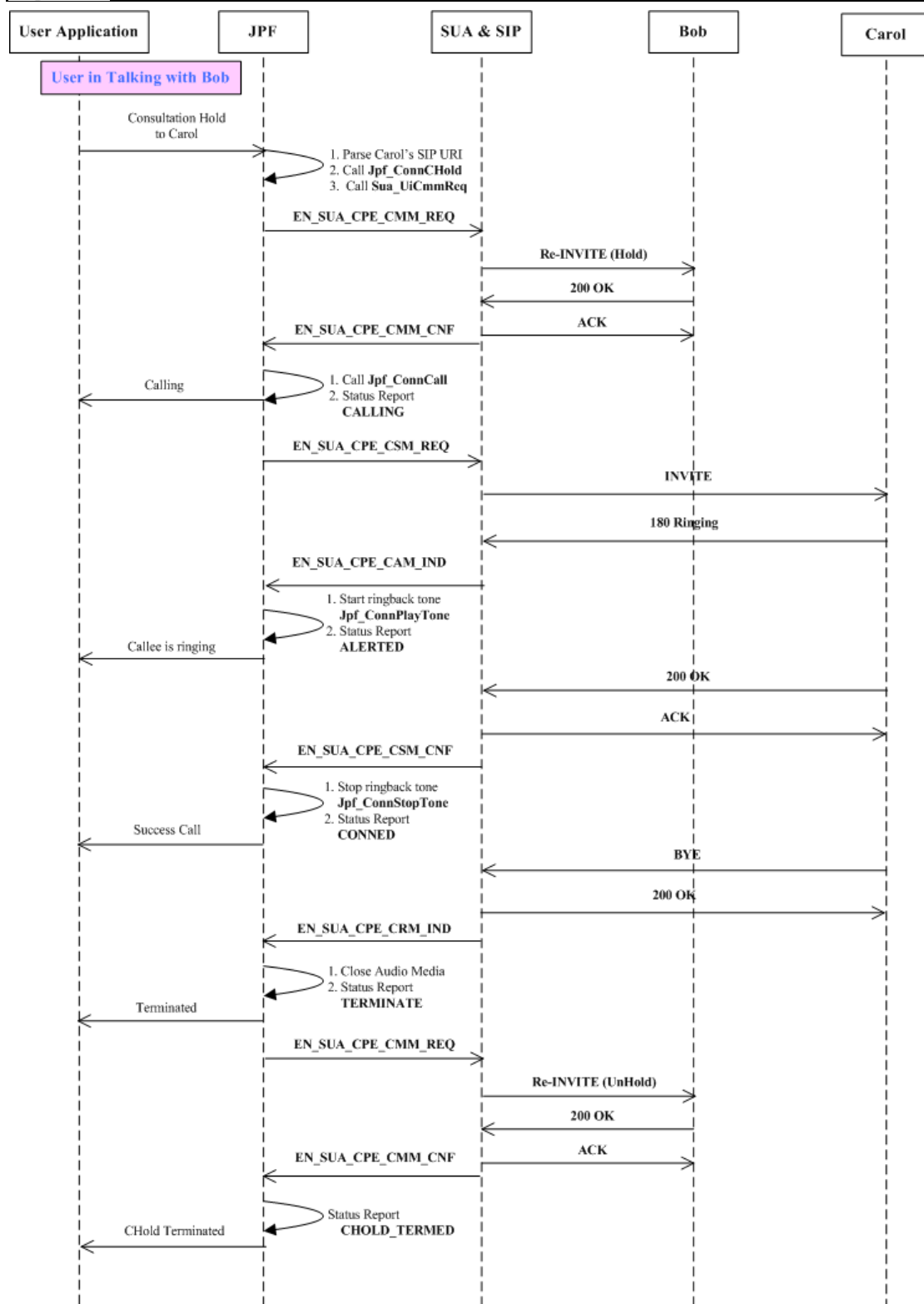


Figure 5-5 Order of Consultation Hold

Environment	Description
Prerequisite	A user endpoint must have been created by Jpf_EndpCreate .

	The user must have established a call with the peer.
Callback function	The user needs to implement the callback function for notifying the connection states. The type of the callback function is PFN_JPFPROCCONNSTAT . When being initialized, this callback function has to be registered by using Jpf_UsrRegCallback .
Data capture	Use the state type having been sent by the callback function to decide the connection state. The user can use Jpf_ConnGetEndpId in the callback function to get the ID of the corresponding endpoint, Jpf_ConnGetCookie to get the user handle, Jpf_ConnGetState to get the connection state, Jpf_ConnGetLocalAddr and Jpf_ConnGetPeerAddr to get the information about the two endpoints of the connection.
Supplement	The user can use interfaces like Jpf_DbSetUsrName and Jpf_DbSetUsrUri to set the user information and use Jpf_DbSetUsedCodecs , Jpf_DbSetAecEnable , Jpf_DbSetAnrEnable , Jpf_DbSetAgcEnable , Jpf_DbSetAsdEnable , Jpf_DbSetLiveToneEnable to set the media capabilities and options.

Table 5-5 User Environments of Consultation Hold

5.6 Unattended Transfer

Unattended Transfer consists of the Transferor providing the Transfer Target's contact to the Transferee. The Transferee attempts to establish a session using that contact and reports the results of that attempt to the Transferor. The signaling relationship between the Transferor and Transferee is not terminated, so the call is recoverable if the Transfer Target cannot be reached.

The process of Unattended Transfer involving the user application, JPF, and other modules are shown below:

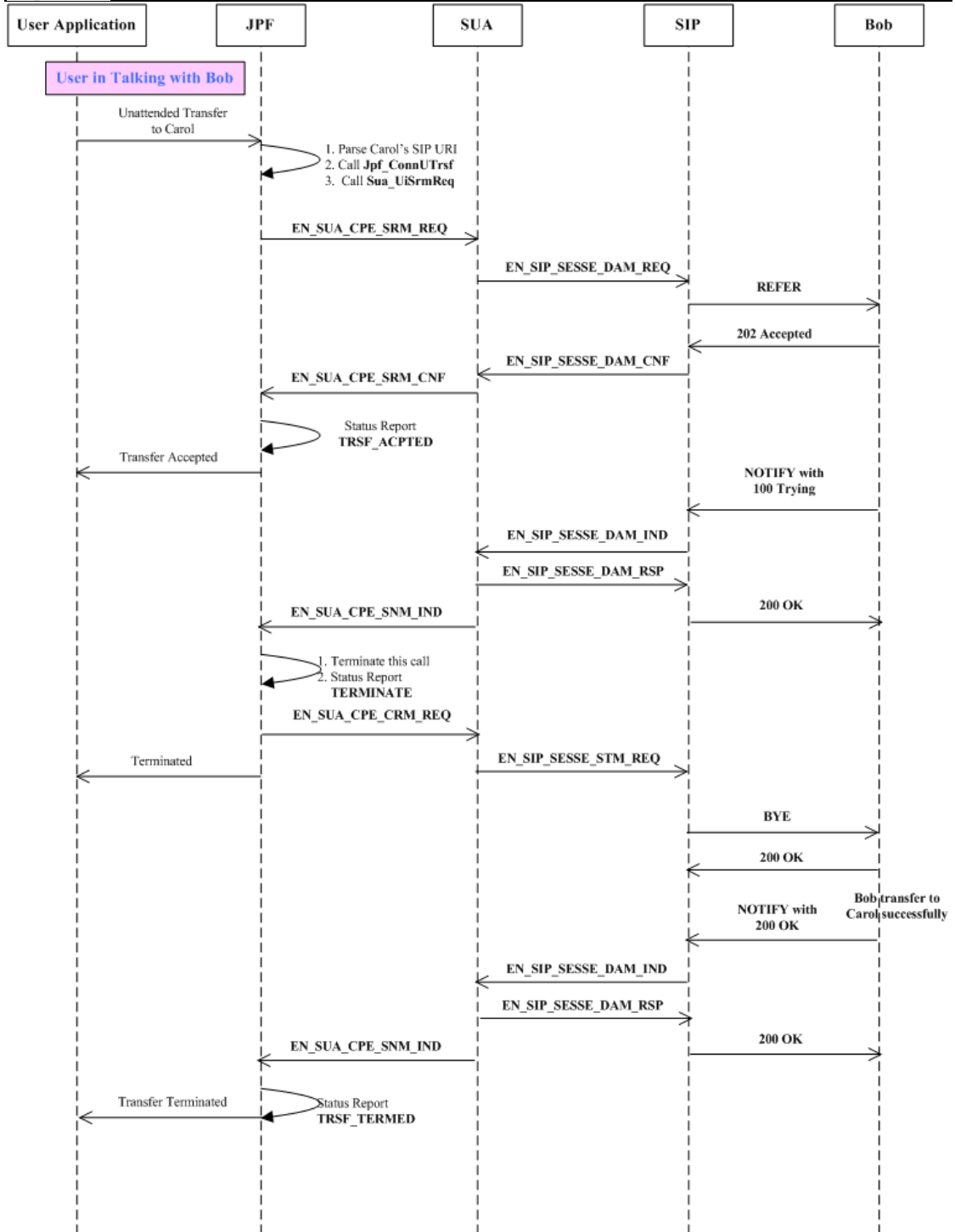


Figure 5-6 Order of Unattended Transfer

Environment	Description
Prerequisite	A user endpoint must have been created by Jpf_EndpCreate . The user must have established a call with the peer.
Callback function	The user needs to implement the callback function for notifying the connection states. The type of the callback function is PFN_JPFPROCCONNSTAT . When

	being initialized, this callback function has to be registered by using Jpf_UsrRegCallback .
Data capture	Use the state type having been sent by the callback function to decide the connection state. The user can use Jpf_ConnGetEndpId in the callback function to get the ID of the corresponding endpoint, Jpf_ConnGetCookie to get the user handle, Jpf_ConnGetState to get the connection state, Jpf_ConnGetLocalAddr and Jpf_ConnGetPeerAddr to get the information about the two endpoints of the connection.
Supplement	The user can use interfaces like Jpf_DbSetUsrName and Jpf_DbSetUsrUri to set the user information and use Jpf_DbSetUsedCodecs , Jpf_DbSetAecEnable , Jpf_DbSetAnrEnable , Jpf_DbSetAgcEnable , Jpf_DbSetAsdEnable , and Jpf_DbSetLiveToneEnable to set media capabilities and options.

Table 5-6 User Environments of Unattended Transfer

5.7 Attended Transfer

Attended Transfer feature allows you to transfer a call to any phone number. You can speak to both parties before connecting. The process of Attended Transfer is shown below:

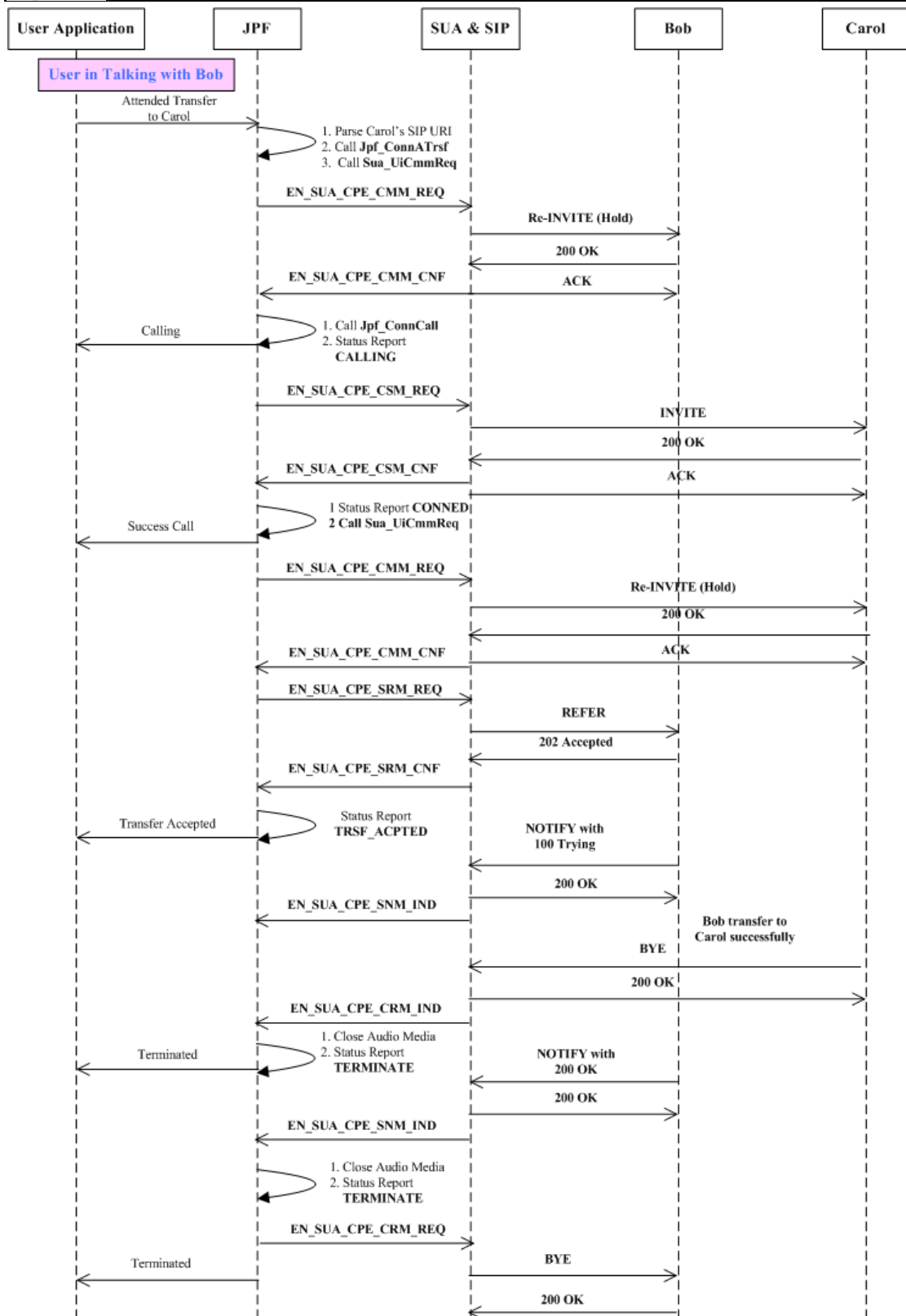


Figure 5-7 Order of Attended Transfer

Environment	Description
Prerequisite	A user endpoint must have been created by Jpf_EndpCreate . The user must have established a call with the peer.
Callback function	The user needs to implement the callback function for notifying the connection states. The type of the callback function is PFN_JPFPROCCONNSTAT . When being initialized, this callback function has to be registered by using Jpf_UsrRegCallback .
Data capture	Use the state type having been sent by the callback function to decide the connection state. The user can use Jpf_ConnGetEndpId in the callback function to get the ID of the corresponding endpoint, Jpf_ConnGetCookie to get the user handle, Jpf_ConnGetState to get the connection state, Jpf_ConnGetLocalAddr and Jpf_ConnGetPeerAddr to get the information about the two endpoints of the connection.
Supplement	The user can use interfaces like Jpf_DbSetUsrName and Jpf_DbSetUsrUri to set the user information and use Jpf_DbSetUsedCodecs , Jpf_DbSetAecEnable , Jpf_DbSetAnrEnable , Jpf_DbSetAgcEnable , Jpf_DbSetAsdEnable , and Jpf_DbSetLiveToneEnable to set the media capabilities and options.

Table 5-7 User Environments of Attended Transfer

5.8 Call Waiting

Call Waiting allows the user to answer a new incoming call when the user is already on a call. The process of Call Waiting is shown below:

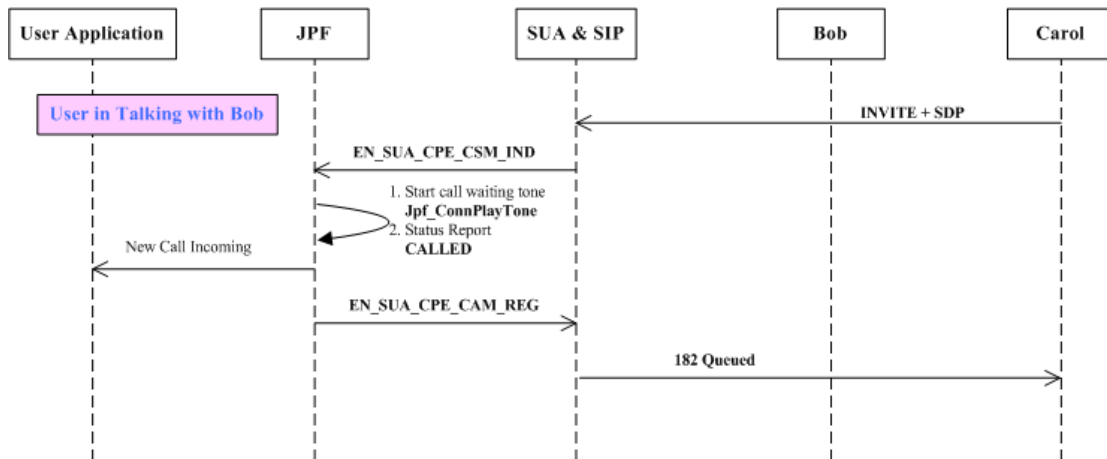


Figure 5-8 Order of Call Waiting

Environment	Description
Prerequisite	A user endpoint must have been created by Jpf_EndpCreate . The user must have established a call with a party.
Callback function	The user needs to implement the callback function for notifying the connection states. The type of the callback function is PFN_JPFPROCCONNSTAT . When

	being initialized, this callback function has to be registered by using Jpf_UsrRegCallback .
Data capture	Use the state type having been sent by the callback function to decide the connection state. The user can use Jpf_ConnGetEndpId in the callback function to get the ID of the corresponding endpoint, Jpf_ConnGetCookie to get the user handle, Jpf_ConnGetState to get the connection state, Jpf_ConnGetPeerAddr and Jpf_ConnGetPeerUri to get the user information of the caller, Jpf_ConnAnswer to accept calls, and Jpf_ConnTerminate to reject calls.
Supplement	The user can use interfaces like Jpf_DbSetUsrName and Jpf_DbSetUsrUri to set the user information.

Table 5-8 User Environments of Call Waiting

5.9 Multiple Line Appearance

Multiple Line Appearance allows the user to handle several calls simultaneously. The process of Multiple Line Appearance is shown below:

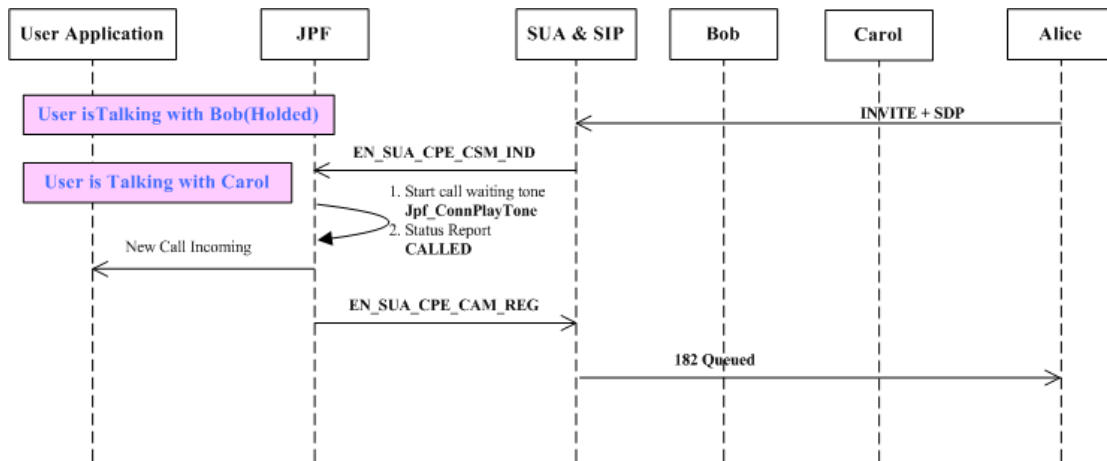


Figure 5-9 Order of Multiple Line Appearance

Environment	Description
Prerequisite	A user endpoint must have been created by Jpf_EndpCreate . The user must have established a call with a party.
Callback function	The user needs to implement the callback function for notifying the connection states. The type of the callback function is PFN_JPFPROCCONNSTAT . When being initialized, this callback function has to be registered by using Jpf_UsrRegCallback .
Data capture	Use the state type having been sent by the callback function to decide the connection state. The user can use Jpf_ConnGetEndpId in the callback function to get the ID of the corresponding endpoint, Jpf_ConnGetCookie to get the user handle, Jpf_ConnGetState to get the connection state, Jpf_ConnGetPeerAddr and Jpf_ConnGetPeerUri to get the user information of the caller, Jpf_ConnAnswer to accept calls, and Jpf_ConnTerminate to reject calls.

Supplement	The user can use interfaces like Jpf_DbSetUsrName and Jpf_DbSetUsrUri to set the user information.
------------	--

Table 5-9 User Environments of Multiple Line Appearance

5.10 Mute

Mute allows the user to set audio states –open or close the audio stream. The process of Mute is shown below:

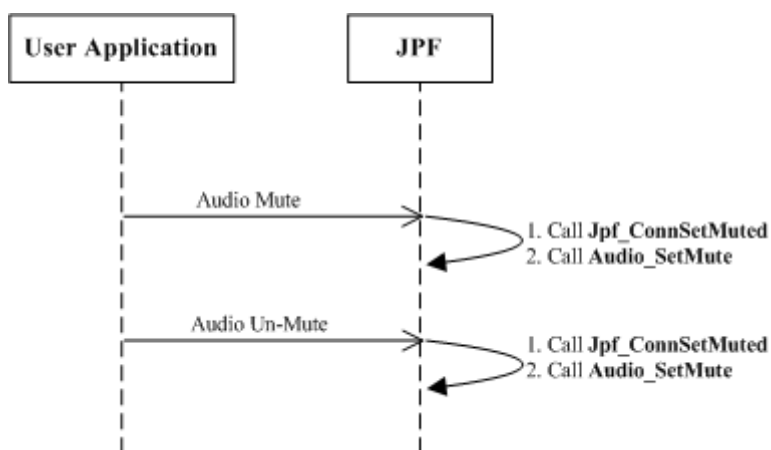


Figure 5-10 Order of Mute

Environment	Description
Prerequisite	A user endpoint must have been created by Jpf_EndpCreate . A connection must have been established, meaning the connection should be in the call out or call in state.
Callback function	The user needs to implement the callback function for notifying the connection states. The type of the callback function is PFN_JPFPROCCONNSTAT . When being initialized, this callback function has to be registered by using Jpf_UsrRegCallback .
Data capture	Use the state type having been sent by the callback function to decide what the connection state is. The state is mainly used to make sure whether an error has occurred when a call was muted, or whether the connection is terminated.
Supplement	The user can use Jpf_ConnGetMuted to get the state indicating whether Mute is on or off.

Table 5-10 User Environments of Mute

5.11 Redial

Redial allows the user to dial the most recent outgoing call again. The information of the most recent outgoing call (the information about the callee of the call) must have been preserved. The way of redialing a call is just the same as making a basic call.

Environment	Description
Prerequisite	A user endpoint must have been created by Jpf_EndpCreate and the information of the latest outgoing call must have been preserved no matter it was successful or not.
Callback function	The user needs to implement the callback function for notifying the connection states. The type of the callback function is PFN_JPFPROCCONNSTAT . When being initialized, this callback function has to be registered by using Jpf_UsrRegCallback .
Data capture	Use the state type having been sent by the callback function to decide the connection state. The user can use Jpf_ConnGetEndpId in the callback function to get the ID of the corresponding endpoint, Jpf_ConnGetCookie to get the user handle, Jpf_ConnGetState to get the connection state, Jpf_ConnGetPeerAddr and Jpf_ConnGetPeerUri to get the user information of the two endpoints of the connection.
Supplement	The user can use interfaces like Jpf_DbSetUsrName and Jpf_DbSetUsrUri to set the user information and Jpf_DbSetUsedCodecs , Jpf_DbSetAecEnable , Jpf_DbSetAnrEnable , Jpf_DbSetAgeEnable , Jpf_DbSetAsdEnable , and Jpf_DbSetLiveToneEnable to set the media capabilities and options.

Table 5-11 User Environments of Redial

5.12 Do Not Disturb

DND allows the user to reject all incoming calls. The process of DND is shown below:

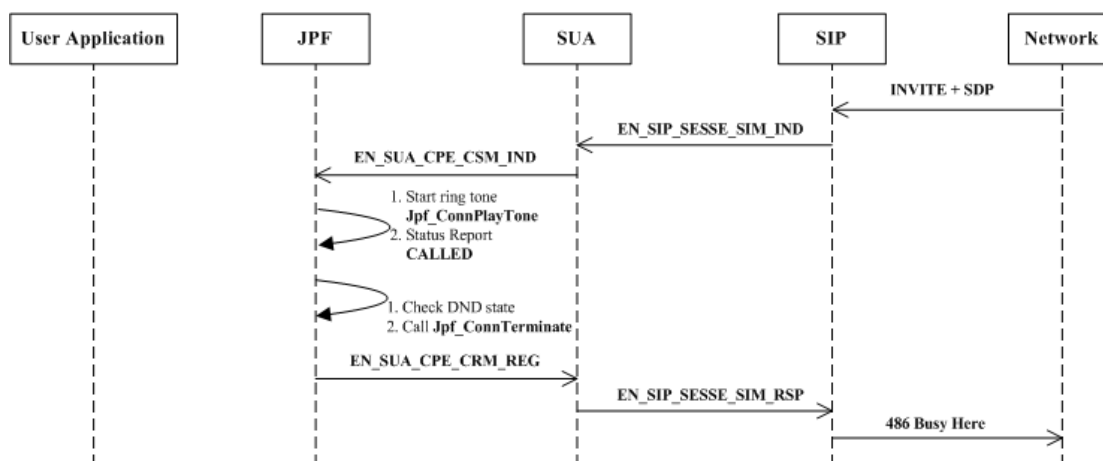


Figure 5-11 Order of Do Not Disturb

Environment	Description
Prerequisite	A user endpoint must have been created by Jpf_EndpCreate .
Callback function	The user needs to implement the callback function for notifying the connection states. The type of the callback function is PFN_JPFPROCCONNSTAT . When being initialized, this callback function has to be registered by using

	Jpf_UsrRegCallback.
Data capture	Use the state type sent by the callback function to decide what the connection state is. If it is a new call, then send a rejection message.
Supplement	The user can use interfaces like Jpf_DbGetDndEnable and Jpf_DbSetDndEnable to set the DND flag.

Table 5-12 User Environments of Do Not Disturb

6. JPF services

JPF provides the following SIP services:

- Register
- Register Refresh
- Un-Register
- Outbound Proxy
- Proxy/WWW Authentication
- Stun
- DTMF

7. Register

If the user sets the registration flag or modifies registration information, then JPF will initiate the registration process. Interfaces for handling registration data are shown below:

Interface	Description
Jpf_DbGetRegDomain	Gets the registration domain name
Jpf_DbGetRegAccount	Gets the registration account
Jpf_DbGetRegPasswd	Gets the password of the registration account
Jpf_DbGetRegEnable	Gets the registration flag
Jpf_DbSetRegDomain	Sets the registration domain name
Jpf_DbSetRegAccount	Sets the registration account
Jpf_DbSetRegPasswd	Sets the password of the registration account
Jpf_DbSetRegEnable	Sets the registration flag

Table 7-1 Interfaces of handling Register related data

The address information at a registrar is kept in the Proxy IP address information. For more information please refer to the function definition document about these interfaces.

When the user has succeeded in registration, SUA will, according to the value of Expire in the EXPIRES Header, activate the timer. When the timer expires, the registration process will be carried out again.

7.1 Un-Register

If the user disables the registration flag, JPF will initiate an Un-Register process to deregister the user account with the registrar

7.2 Outbound Proxy

If the user sets the Proxy server information, then all the user call services will be completed through the Proxy server. The Proxy data operation interfaces are shown below:

Interface	Description
Jpf_DbGetProxyIp	Gets the IP address of a Proxy server
Jpf_DbGetProxyUdpPort	Gets the SIP UDP port number of a Proxy server
Jpf_DbGetProxyTcpPort	Gets the SIP TCP port number of a Proxy server
Jpf_DbSetProxyIp	Sets the IP address of a Proxy server
Jpf_DbSetProxyUdpPort	Sets the SIP UDP port number of a Proxy server
Jpf_DbSetProxyTcpPort	Sets the SIP TCP port number of a Proxy server

Table 7-2 Proxy Data Operation Interfaces

7.3 Proxy/WWW Authentication

After sending a REGISTER or INVITE message, the user may receive a 401 or 407 response indicating that the Proxy server needs to authenticate the user identity. Then the user needs to construct a new request message to send the user account and password credential to the Proxy server.

7.4 STUN

If the user is behind a firewall or NAT, then the user can use the STUN switch to perform NAT traversal. The STUN data operation interfaces are **Jpf_DbGetStunEnable** and **Jpf_DbSetStunEnable**.

7.5 DTMF

The user can use interface **Jpf_ConnDtmf** to send outbound (RFC2833) or inbound DTMF signals. The user has to set the callback function in the AUDIO layer to receive DTMF signals. For more information please refer to RFC 2833.