Introducing Natural RPC

This section covers the following topics:

- General Information
- Natural RPC Operation in Non-Conversational Mode
- Natural RPC Operation in Conversational Mode
- Conversational versus Non-Conversational Mode
- Database Transactions
- Location of Conversations
- Natural RPC Terminology

General Information

- Purpose
- Advantages of Natural Remote Procedure Calls
- Natural RPC Modes of Operation
- Availability on Various Platforms
- Support of Non-Natural Environments (EntireX RPC)

Purpose

The Natural RPC facility enables a client Natural program to issue a `CALLNAT` statement to invoke a subprogram in a server Natural. The Natural client and server sessions may run on the same or on a different computer. For example, a Natural client program on a Windows computer can issue a `CALLNAT` statement against a mainframe server in order to retrieve data from a mainframe database. The same Windows computer can act as a server if a Natural client program running under, for example, UNIX issues a `CALLNAT` statement requesting data from this server Natural.

Advantages of Natural Remote Procedure Calls

Natural RPC exploits the advantages of client server computing. In a typical scenario, Natural on a Windows client computer accesses server data (using a middleware layer) from a Natural on a mainframe computer. The following advantages arise from that:

- The end user on the client side can use a Natural application with a graphical user interface.
- A large database can be accessed on a mainframe server.

- Network traffic can be minimized when only relevant data are sent from client to server and back.

Natural RPC Modes of Operation

The Natural Remote Procedure Call offers the following modes of operation:

- non-conversational mode (in the following texts this mode is meant unless otherwise specified)
- conversational mode

These modes are described in detail in the following sections. For a comparison of the advantages and disadvantages of these modes, refer to *Conversational versus Non-Conversational Mode*.

Availability on Various Platforms

You can use the Natural RPC on various platforms under the following operating systems:

Mainframe Environments

- z/OS
- z/VSE
- VM/CMS
- BS2000/OSD

Natural RPC on mainframes is supported under the following TP monitors:

- Com-plete
- CICS
- IMS TM
- TSO
- UTM

Also, it is available in batch mode.

Other Environments

- Windows
- UNIX
- OpenVMS

On all of these platforms, Natural can act as both client and server.

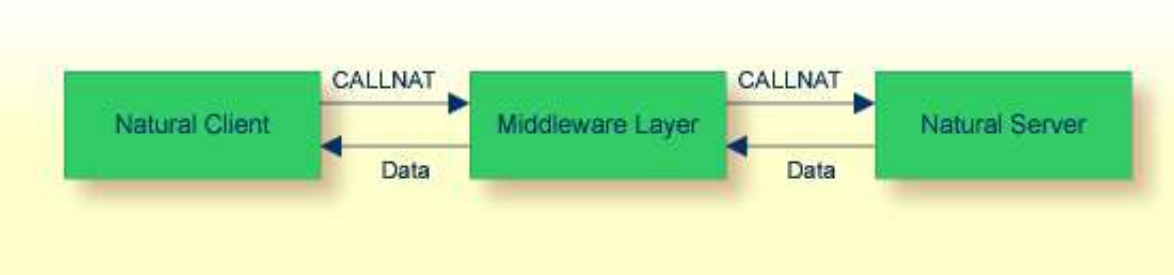
Support of Non-Natural Environments (EntireX RPC)

Non-Natural environments (3GL and other programming languages) are supported on the client and the server side. Thus, a non-Natural client can communicate with a Natural RPC server, and a Natural client can communicate with a non-Natural RPC server. This is enabled by the use of the EntireX RPC.

Natural RPC Operation in Non-Conversational Mode

The non-conversational mode should be used only to accomplish a single exchange of data with a partner. See also Conversational versus Non-Conversational Mode.

The Natural RPC technique uses the Natural statement `CALLNAT`, so that both local and remote subprogram calls can be issued in parallel. Remote program calls work synchronously. As a remote procedure call, a `CALLNAT` would, simply speaking, take the following route:



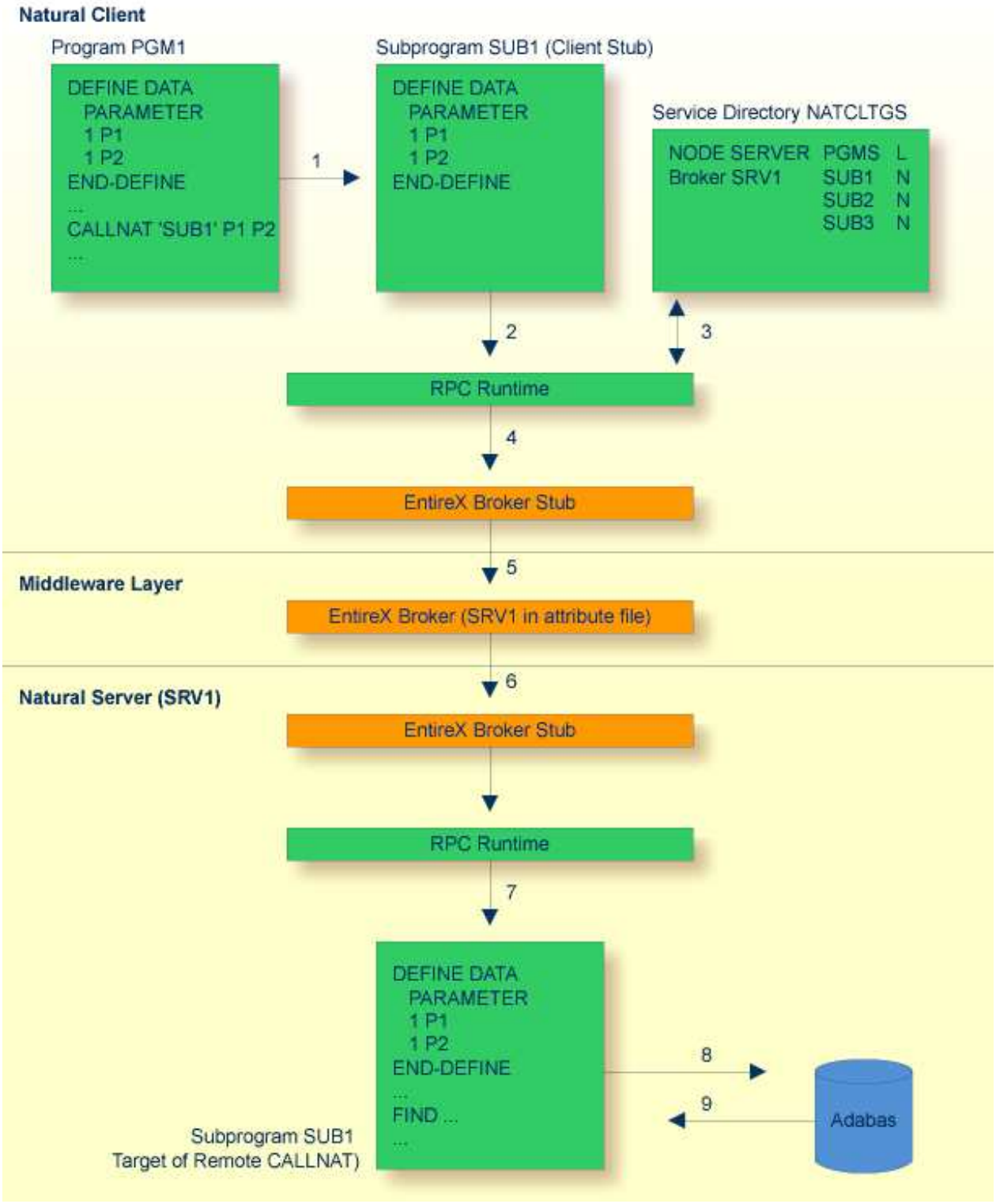
The `CALLNAT` issued from the Natural client is routed via a middleware layer to the Natural server which passes data back to the client.

Usually, the middleware layer consists of the Software AG product EntireX Broker which uses the ACI protocol. EntireX Broker uses either Entire Net-Work or TCP/IP as communication layer.

A detailed example of the RPC control flow is described below.

Issuing CALLNATs in an RPC Environment

`CALLNAT` control flow details in a remote procedure are illustrated below. For greater clarity, the return path is not shown, but it is analogous; the numbers refer to the description:



1. From the Natural client, the program PGM1 issues a CALLNAT to the subprogram SUB1. PGM1 does not know if its CALLNAT will result in a local or in a remote CALLNAT.

   As the target SUB1 resides on a server, the CALLNAT accesses a stub subprogram (interface object) SUB1 instead. This client stub subprogram has been created automatically or manually (by using the SYSRPC utility's stub generation (SG) function).

The stub has the same name as the target subprogram and contains parameters identical with those used in program PGM1 and in the target subprogram SUB1 on the server. It also contains control information used internally by the RPC.

If the AUTORPC profile parameter is set to ON and Natural cannot find the subprogram in the local environment, Natural interprets this as a remote procedure call and generates the parameter data area (PDA) dynamically during runtime.

Natural also tries to find this subprogram in the service directory NATCLTGS.

For further information on the SYSRPC stub generation function, see *Creating Stub Subprograms*.

If you want to work without stubs, see *Working with Automatic Natural RPC Execution*.

2. The stub then sets up a CALLNAT to an RPC client service routine.
3. The client RPC runtime checks in the service directory NATCLTGS on which node and server the CALLNAT is to be performed and whether a logon is required.

   The CALLNAT data including the parameter list and, if required, the logon data are passed to a middleware layer.
4. In this example, this middleware layer consists of the Software AG product EntireX Broker. Therefore, the CALLNAT data is first passed to an EntireX Broker stub on the client.
5. From the EntireX Broker stub, the CALLNAT data is passed to the EntireX Broker. The EntireX Broker is a product that can reside:

   - on the client computer
   - on the server computer or
   - on a third platform.

   For the data to be passed on successfully, the server SRV1 must be defined in the EntireX Broker attribute file and SRV1 must be already up, thus having registered with EntireX Broker.

   For information on how to define servers in the EntireX Broker attribute file, see the EntireX Broker documentation.
6. From the middleware layer, the CALLNAT data is passed on to the EntireX Broker Stub on the Natural Server platform and from there to the RPC server service routine.

   The RPC server service routine validates the logon data (if present) and performs a logon (if requested).
7. The RPC server service routine invokes the target subprogram SUB1 and passes the data, if requested.

   At this point, the target subprogram SUB1 has all the required data to execute just as if it had been invoked by a local program PGM1.

8. Then, for example, the subprogram SUB1 can issue a FIND statement to the server's Adabas database. SUB1 does not know whether it has been started by a local or by a remote CALLNAT.

9. Adabas FINDs the data and passes them to SUB1.

   Then, SUB1 returns the Adabas data to the calling server service routine. From there, it is passed it back to PGM1 via the middleware layer. It takes the same route as described in Steps 1 to 8, but in reverse order.

Natural RPC Operation in Conversational Mode

A conversational RPC is a static connection of limited duration between a client and a server. It provides a number of services (subprograms) defined by the client, which are all executed within one server task that is exclusively available to the client for the duration of the conversation. It is implemented in a program using an OPEN CONVERSATION statement and a CLOSE CONVERSATION statement.

Multiple connections (conversations) can exist at the same time. They are maintained by the client by means of conversation IDs, and each of them is performed on a different server. Remote procedure calls which do not belong to a given conversation are executed on a different server, within a different server task.

During a conversation, you can define and share a data area called context area between the remote subprograms on the server side. For further information, see *Defining Context Variables for Natural RPC* in the Natural *Statements* documentation.

A conversation may be local or remote.

Example:

```
OPEN CONVERSATION USING SUBPROGRAM 'S1''S2'
     CALLNAT 'S1' PARMS1
     CALLNAT 'S2' PARMS2
CLOSE CONVERSATION ALL
```

Both subprograms (S1 and S2) must be accessed at the same location, that is, either locally or remotely. It is not admissible to mix up local and remote CALLNATs within a conversation. If the subprograms are executed remotely, both subprograms will be executed by the same server task.

Analogously to non-conversational RPC CALLNATs, conversations may first be written and tested locally and can then be transferred to the servers.

General Rules for Local/Remote Subprogram Execution

Local Subprogram Execution

If you execute subprograms locally, the following rule applies:

- A subprogram may not call another subprogram which is a member of the conversation.

Other subprograms not listed in the OPEN CONVERSATION statement may be called. They are however executed in non-conversational mode.

Remote Subprogram Execution

If you execute subprograms remotely, the following rule applies:

- A subprogram `S1` may call another subprogram `S2` which is a member of the conversation.

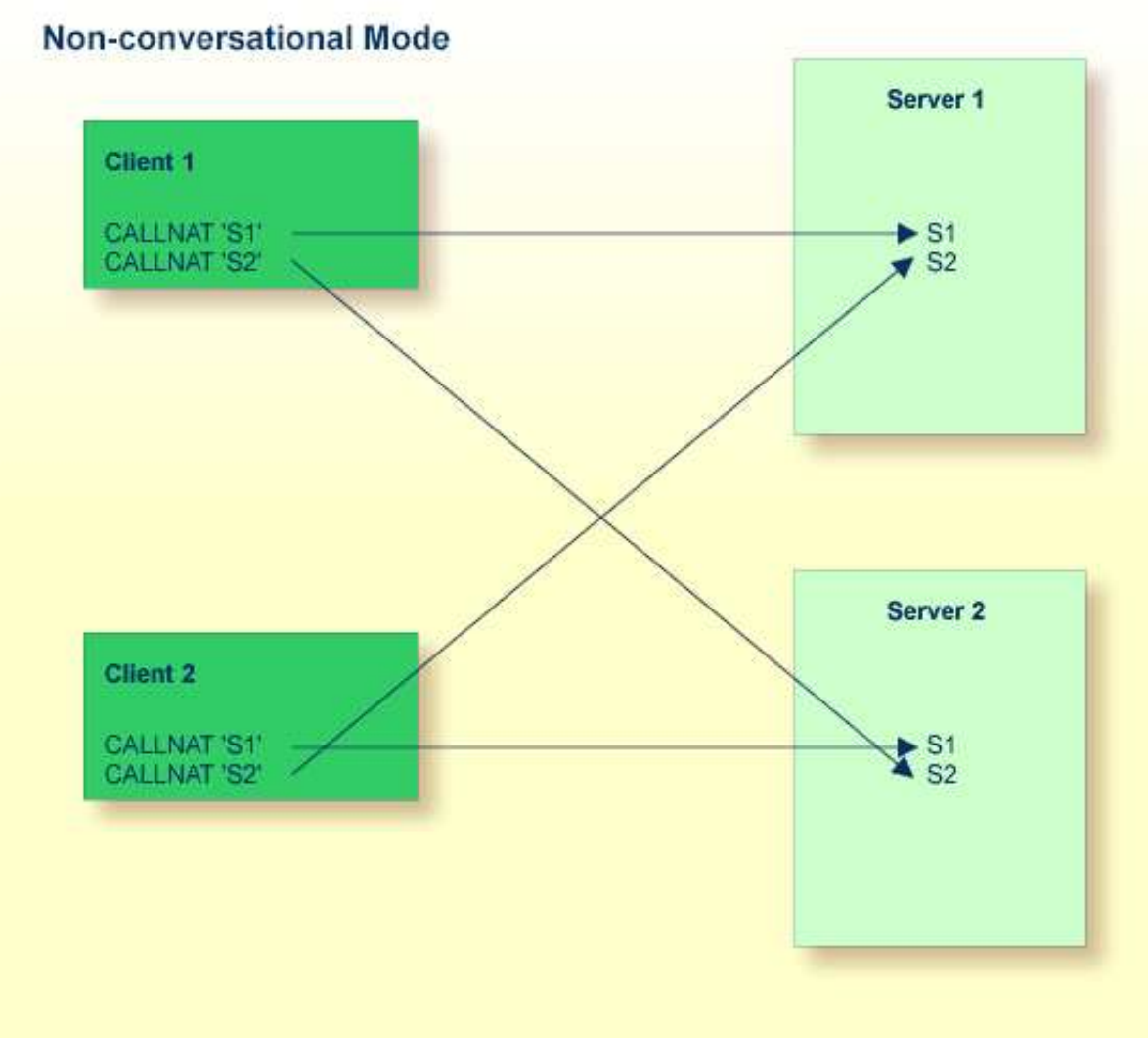
This `CALLNAT` will be executed in non-conversational mode because it was invoked indirectly. Thus, the subprogram S2 does not have access to the context area.

Conversational versus Non-Conversational Mode

In a client-server environment where several clients access several servers in non-conversational mode, there may be the problem that identical `CALLNAT` requests from different clients are executed on the same server.

This means, for example, that a `CALLNAT 'S1'` from Client 1 executes Subprogram `S1` on Server 1 (`S1` is writing a record to the database). The transaction for Client 1 is not yet complete (no `END TRANSACTION`) when Client 2 also sends a `CALLNAT 'S1'` to Server 1, thus overwriting the data from Client 1. If Client 1 then sends a `CALLNAT 'S2'` (meaning `END TRANSACTION`), Client 1 supposes its data have been saved correctly, although in fact the data from Client 2's identical `CALLNAT` were saved.

The diagram below illustrates this with two clients and two servers. In such a scenario, you cannot control whether two identical `CALLNAT`s from two different clients access the same subprogram on the same server:

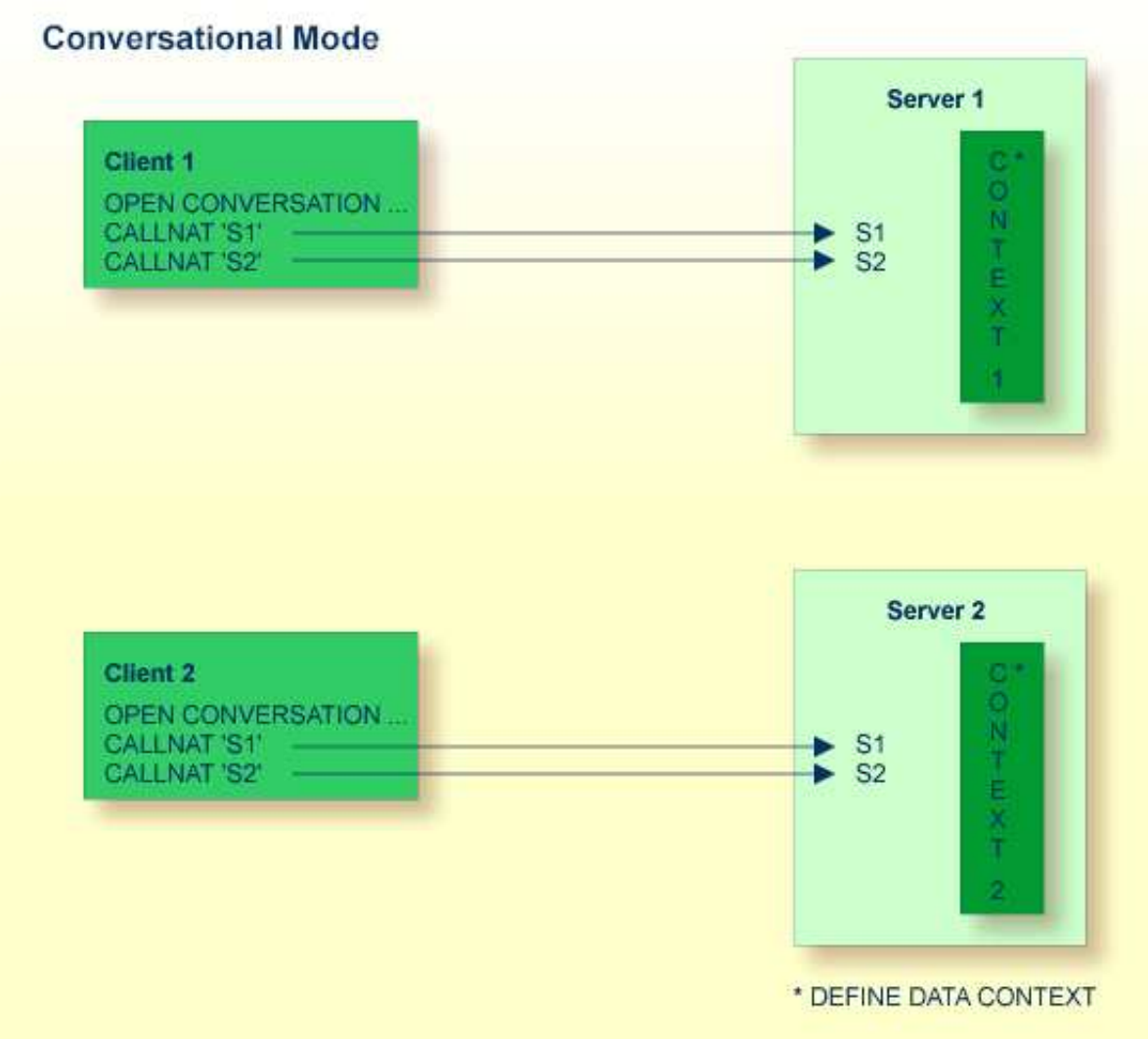


In the above example, `CALLNAT 'S2'` from Client 1 can access subprogram `S2` on Server 1 and on Server 2. `CALLNAT 'S2'` from Client 2 has the same choice.

Similarly, `CALLNAT 'S1'` from Client 1 could access Subprogram `S1` on Server 1 and on Server 2, while `CALLNAT 'S1'` from Client 2 has the same choice.

It is obvious that interference can be a problem here if the subprograms are designed to be executed within one server task context.

You can avoid the potential problems of a non-conversational RPC by defining a more complex RPC transaction in conversational mode:



You do this by opening a conversation. This involves the use of the `OPEN CONVERSATION` statement on the client side, referring to `CALLNAT 'S1'` and `CALLNAT 'S2'`. Opening such a conversation reserves one entire server task (for example, Server 1) and no other remote `CALLNAT`s may interrupt this conversation on this server before this conversation has been closed. In addition, you can define a common context area for the two subprograms on the server side by using the `DEFINE DATA CONTEXT` statement.

General Rules for Use of Conversational/Non-Conversational RPC

As a general rule, the following applies:

- Use the **conversational RPC** to ensure that a defined list of subprograms is executed exclusively within one context.
- Use the **non-conversational RPC** if each of your subprograms can be used within a different server task or if the transaction does not extend over more than one server call. The advantage of this is that no server blocks over a significant amount of time and you only need a relatively small number of server tasks.

Possible Disadvantage of Using Conversational RPC

A possible disadvantage of conversational RPCs is that you reserve an entire server task, thus blocking all other subprograms on this server. As a consequence, other CALLNATs might have to wait or more server tasks must be started.

Database Transactions

The database transactions on the client and server sides run independent of each other. That is, an END TRANSACTION or BACKOUT TRANSACTION executed on the server side does not influence the database transaction on the client side and vice-versa.

At the end of each non-conversational CALLNAT and at the end of each conversation, an implicit BACKOUT TRANSACTION is executed on the server side. To commit the changes made by the remote CALLNAT(s), you have the following options:

Non-conversational CALLNAT

1. Execute an explicit END TRANSACTION before leaving the CALLNAT.
2. Set the Natural profile parameter ETEOP to ON. This results in an implicit END TRANSACTION at the end of each non-conversational CALLNAT.

   Depending on the setting of the parameter SRVCMIT, the END TRANSACTION is executed either before the reply is sent to the client (SRVCMIT=B) or after the reply has been successfully sent to the client (SRVCMIT=A). SRVCMIT=B is the default and is compatible with earlier versions of the RPC.

Conversational CALLNAT

1. Execute an explicit END TRANSACTION on the server before the conversation is terminated by the client
2. Set the Natural profile parameter ETEOP to ON. This results in an implicit END TRANSACTION at the end of each conversation.

   Depending on the setting of the parameter SRVCMIT, the END TRANSACTION is executed either before the reply is sent to the client (SRVCMIT=B) or after the reply has been successfully sent to the client (SRVCMIT=A). SRVCMIT=B is the default and is compatible with earlier versions of the RPC.
3. Before executing the CLOSE CONVERSATION statement, call the application programming interface USR2032N on the client side. This will cause an implicit END TRANSACTION at the end of the individual conversation.

Location of Conversations

Both subprograms S1 and S2 (shown in the figure above) must be accessed at the same location, i.e. either locally or remotely. You may not mix up local and remote CALLNATs within a conversation. If the subprograms are executed remotely, both subprograms will be executed by the same server task.

Natural RPC Terminology

The following table provides an overview of important key terms used in the SYSRPC Utility and the Natural RPC documentation:

Term	Explanation
Client Stub	Accepts the CALLNAT requests on the client side, marshalls the parameters passed, transmits the data through the Natural RPC runtime and the transport layer to the remote server, unmarshalls the result and returns it to the caller. The client stub is the local subprogram via which the server subprogram is called. The client stub has the same name and contains the same parameters as the corresponding server subprogram.
EntireX Broker Stub	Interface between the Natural RPC runtime and the EntireX Broker transport layer which exchanges marshalled data between client and server.
Impersonation	Impersonation assumes that access to the operating system on which a Natural RPC server is running is controlled by an SAF-compliant external security system. User authentication is performed by this external security system. Impersonation means that after the authentication has been successful and the user's identity is established, any subsequent authorization checks will be performed based on this identity. This includes authorization checks for access to external resources (for example, databases or work files). After successful authentication the user cannot "change his/her identity", that is, he/she cannot use a different user ID. See *Impersonation* in *Using Security*.
Interface Object	In earlier versions of EntireX, the term "stub" was also used to refer to application-dependent, Workbench-generated pieces of code for issuing and receiving remote procedure calls. These objects are now referred to as *interface objects*.
NATCLTGS	The name of the Natural subprogram generated with the SYSRPC utility to implement the service directory (see below).
Node Name	The name of the node to which the remote CALLNAT is sent. In case of communication via the EntireX Broker, the node name is the name of the EntireX Broker for example, as defined in the EntireX Broker attribute file, in the field BROKER-ID.
RPC Parameters	All parameters available to control a Natural RPC are described in detail in the Natural Parameter Reference documentation. See the section *Profile Parameters*.
SERVDIRX	The name of the XML-formatted file (Natural text member) generated with the SYSRPC utility to implement the service directory (see below).
Service Directory	The service directory contains information on the services (subprograms) that a server provides. It can be locally available on each client node, or it can be located on a remote directory server referenced by the profile parameterRDS.

Server Name	The name of the server on which the CALLNAT is to be executed. In case of communication via EntireX Broker, the server name is the name as defined in the EntireX Broker attribute file in the field SERVER.
Server Task	A Natural task which offers services (subprograms). This is typically a batch task or asynchronous task. It is identified by a server name.