BEA eLink Adapter for Mainframe

Samples Guide

BEA eLink Adapter for Mainframe 4.0
Document Edition 4.0
January 2001

Copyright

Copyright © 2001 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, ObjectBroker, TOP END, and Tuxedo are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Connect, BEA Manager, BEA MessageQ, BEA Jolt, M3, eSolutions, eLink, WebLogic, WebLogic Enterprise, WebLogic Commerce Server, and WebLogic Personalization Server are trademarks of BEA Systems, Inc.

All other company names may be trademarks of the respective companies with which they are associated.

BEA® eLink™ Adapter for Mainframe User Guide

Document Edition	Part Number	Date	Software Version
4.0		January 2001	BEA eLink Adapter for Mainframe, 4.0

Contents

1 ATMI to CPI-C Function Mapping

ATMI Calls Mapped to CPI-C Verbs

This appendix lists the most common ATMI function calls and shows how their parameters map to CPIC verbs. The mappings are shown in the following order:

- tpcall()
- tpacall() with or without reply
- tpgetrply()
- tpservice()
- tpreturn()
- tpcancel()
- tpconnect()
- tpsend()
- tprecv()
- tpdiscon()
- tpforward ()

The tables on the following pages show the parameters of the ATMI call, what the content or meaning of the parameters is, and notes that indicate the usage with the CPIC verbs.

Table 0-1 tpcall

tpcall()	Parameters	Contents	CPIC Notes
svc		Service Name	Used in CMALLC to identify the CICS transaction to be invoked.
idata		User data	This data is sent in CMSENDs until completely transmitted
len		Length of User data	0 < data <= 32K
odata		Reply data	CMRCV receives the data until it has been completely transmitted (data_received is set to CM_COMPLETE_DATA_RECEIVED) and return code is set to CM_OK or CM_DEALLOCATE_NORMAL
olen		Reply data length	0 < data < 32K
flags	TPNOTRAN	Not part of a transaction	
	TPNOCHANGE	N/A	Local
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local

Table 0-2 tpacall

tpacall()	Parameter	Contents	CPIC Notes
svc		Service Name	Used in CMALLC to identify the CICS transaction to be invoked.
data		User data	This data is sent in CMSENDs until completely transmitted

Table 0-2 tpacall

tpacall()	Parameter	Contents	CPIC Notes
len		Length of user data	0 < data <= 32K
flags	TPNOREPLY	false	The last data is sent with a CMSEND with send_type set to CMSEND_AND_PREP_TO_RECEIVE. This changes the state of the conversation to receive and a CMRCV is issued to await the reply
		true	Since no reply is expected, a CMDEAL deallocates the conversation after all data has been received
	TPNOTRAN	Not part of a transaction	Sync level 2 not supported in this release, therefore the service call should not be in transaction mode
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local

Table 0-3 tpgetrply

tpgetrply()	Parameters	Contents	CPIC Notes
cd		call descriptor	The call descriptor is mapped to the CONVID returned by the CMINIT when the LU6.2 was initiated
data		User data	Data received from CMRCV if WHAT_RECEIVED set to DATA_COMPLETE
len		Length of user data	0 < data <= 32K

Table 0-3 tpgetrply

tpgetrply()	Parameters	Contents	CPIC Notes
flags	TPGETANY	If true, data is returned from any conversation. If false, data is returned from conversation associated with the cd	Data available on any conversation is returned to the requestor
	TPNOCHANGE	Local to the requestor	Limited buffer types supported.
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local

Table 0-4 tpservice

tpservice()	Parameters	Contents	CPIC Notes
svcinfo		Service information and data	User Data captured from a CMRCV populates the TPSVCINFO structure user data area. Service characteristics are obtained from the service attributes in the DMCONFIG and UBBCONFIG files.
name		Service name	The service name associated with the 8 character RNAME sent from CICS.
data		User data	Data captured from CMRCV
len		Length of user data	0 < data <= 32K Length of data received
cd		call descriptor	The call descriptor associated with the CONVID returned by the CMINIT when the LU6.2 was initiated
appkey		32-bit key (if used)	For security
cltid		set by BEA Tuxedo	For security

Table 0-4 tpservice

tpservice()	Parameters	Contents	CPIC Notes
flags	TPCONV	If true, service is conversational	
	TPTRAN	N/A	Sync level 2 not supported in this release, therefore the service cannot be called in transaction mode
	TPNOREPLY	If true, requestor not expecting a reply	The conversation is terminated with a CMDEAL normal
	TPSENDONLY	N/A	If set, the CPIC conversation in CICS should be in receive state. If not set, the CICS CPIC conversation state will be in send state.
	TPRECVONLY	N/A	If set, the CPIC conversation in CICS remains in send state.

Table 0-5 tpreturn

tpreturn()	Parameters	Contents	CPIC Notes
rval		TPSUCCESS	Set to TPSUCCESS when conversation terminates with a normal deallocation.
		TPSVCERR	Set to TPESVCERR when the conversation has terminated with a non-normal deallocation type or other error.
rcode		Set by the application	N/A
data		User data	Data is returned to the CICS transaction from a successful CMRCV with data received set to CM_DATA_COMPLETE and return code of CM_DEALLOCATE_NORMAL. If the service fails, no data is returned to the caller and the conversation is deallocated abnormally.
len		Length of data returned	0 < data <= 32K
flags		N/A	N/A

Table 0-6 tpcancel

tpcancel()	Parameters	Contents	CPIC Notes
cd		The connection descriptor on which a `tpgetreply()` is waiting.	CMDEAL abnormal is issued on the conversation with CONVID mapped from call descriptor

Table 0-7 tpconnect

tpconnect()	Parameters	Contents	CPIC Notes
svc		The local service name representing the service to be invoked. in CICS	The name is used to find the RNAME. The RNAME should match the TPName in CICS and will be used by CMINIT and CMALLC to initiate and allocate the conversation
data		User data	This data is sent in CMSENDs until completely transmitted
len		Length of User data	0 < data <= 32K
flags	`TPNOTRAN`	True	Sync level 2 not supported in this release, therefore the service call should not be in transaction mode
	`TPSENDONLY`	If true, the conversation stays in or changes to send state	The conversation remains in send state. This is the default.
	`TPRECVONLY`	If true, the conversation stays in or changes to receive state	Immediately after the allocate BEA Tuxedo sends a CMSEND with no data and send_type set to CM_SEND_AND_PREP_TO_RECEIVE
	`TPNOBLOCK`	N/A	Local
	`TPNOTIME`	N/A	Local
	`TPSIGRSTRT`	N/A	Local

Table 0-8 tpsend

tpsend()	Parameters	Contents	CPIC Notes
cd		The connection descriptor	This locally assigned connection descriptor has been mapped to the CONVID returned in the CMINIT and CMALLC on behalf of the `tpconnect()`
data		User data	ASCII/EBCDIC conversion may be required before sending to CICS. See Chapter 5 “Buffers”
len		Length of User data	0 < data <= 32K
flags	`TPRECVONLY`	If true, the conversation changes to receive state	The state of the conversation changes from send to receive. A CMSEND is sent with send_type set to CM_SEND_AND_PREP_TO_RECEIVE
	`TPNOBLOCK`	N/A	Local
	`TPNOTIME`	N/A	Local
	`TPSIGRSTRT`	N/A	Local
revent	`TPEV_DISCONIMM`	If set, the LU6.2 conversation has been terminated abnormally	If the return code from a CMRCV is deallocate_abnormal, the conversation is terminated. A disconnect event is sent to the sending process.
	`TPEV_SVCERR`	If set, the LU6.2 conversation has been terminated abnormally	Any return code other than CM_OK or CM_DEALLOCATE_NORMAL is treated as a TPEV_SVCERR
	`TPEV_SVCFAIL`	If set, the LU6.2 conversation has been terminated abnormally	If the return code from CMRCV is CM_TP_NOT_AVAIL_NO_RETRY or CM_TP_RESOURCE_FAILURE_NO_RETRY, `revent` is set to TPEV_SVCFAIL

Table 0-9 tprecv

tprecv()	Parameters	Contents	CPIC Notes
cd		The connection descriptor	This locally assigned connection descriptor has been mapped to the CONVID returned in the CMINIT and CMALLC issued by the initiator of this conversation
data		User data	Date to be received using a CMRCV_immediate and returned to the BEA Tuxedo service
len		Length of User data	0 < data <= 32K
flags	TPNOCHANGE	Local	Must be a supported buffer type
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local
revent	TPEV_DISCONIMM	If set, the LU6.2 conversation has been terminated abnormally	If the return code from a CMSEND is deallocate_abnormal, the conversation is terminated. A disconnect event is sent to the sending process.
	TPEV_SENDONLY	If set, the LU6.2 conversation changes to send if partner allows it	The sending partner has sent a CMSEND with send_type set to CM_SEND_AND_PREP_TO_RECEIVE
	TPEV_SVCERR	If set, the LU6.2 conversation has been terminated abnormally	Any return code other than CM_OK or CM_DEALLOCATE_NORMAL is treated as a TPEV_SVCERR
	TPEV_SVCFAIL		If the return code from CMRCV is CM_TP_NOT_AVAIL_NO_RETRY or CM_TP_RESOURCE_FAILURE_NO_RETRY, revent is set to TPEV_SVCFAIL
	TPEV_SVCSUCC	If set, the conversation has completed normally	The return code from CMRCV was set to CM_DEALLOCATE_NORMAL. This indicates that the sending TP has completed and deallocated the conversation normally.

Table 0-10 tpdiscon

tpdiscon() Parameters	Contents	CPIC Notes
cd	The connection descriptor	This connection descriptor in mapped to the CONVID returned from CMINIT or CMACCP to the originator of the conversation

Table 0-11 tpforward

tpforward () Parameters	Contents	CPIC Notes
svc	Service name	tpforward() is treated as if it were a `tpacall()`. A CMINIT and subsequent CMALLC are issued to initialize and allocate a session for a conversation. ClientID must be propagated to the CICS transaction in a TPSVCINFO record
data	User data	Data is sent using CMSEND. The last CMSEND is sent with send_type of deallocate_normaL
len	Length of data returned	0 < data <= 32K
flags	Refer to `tpacall`()	

2 CPI-C Parameters and Values

CPI-C Verbs

The following tables show the parameters required for each CPI-C verb and the permissible values when used with BEA eLink Adapter for Mainframe software.

Parameter	In/Out	Value	Usage
`conversation_ID`	Output	Returned by the SNA Stack	Used on every other call
`sym_dest_name`	Input	Symbolic name that represents the partner LU, the modename, and the TP (Tuxedo service)	If blank, SET calls must be made to initialize conversation parameters
`return code`	Output	See list of CPI-C return codes	If not CM_OK, conversation fails.

Parameter	In/Out	Value	Usage
`conversation_ID`	Input	Value returned from CMINIT	
`return_code`	Output	See list of CPI-C return codes	If not CM_OK, or CM_ALLOCATION_FAILURE_RETRY, conversation fails

Parameter	In/Out	Value	Usage
conversation_ID	Input	Value returned from CMINIT	
buffer	Input	Name of variable to be populated by data received	
requested_length	Input	Name of variable where amount of data to be received is posted	0 <= requested_length < 32K
data_received	Output	CM_COMPLETE	Indicates enough data in LU buffer to satisfy requested length
		CM_INCOMPLETE	Another CMRCV must be issued to get enough data to satisfy the requested length
received-length	Output	Name of variable where amount of data returned is posted	Returned data may be shorter than requested amount including zero
status_received	Output	CM_NO_STATUS	No state change.
		CM_SEND_RECEIVED	State of the conversation is changed to send
request_to_send_received	Output	CM_REQ_TO_SEND_RECEIVED	Receiver enters send state.
		CM_REQ_TO_SEND_NOT_RECEIVED	Partner has not requested a state change
return_code	Output	See list of CPI-C return codes	If not CM_OK or CM_DEALLOCATE_NORMAL, the conversation is deallocated.

Parameter	In/Out	Value	Usage
conversation_ID	Input	Value returned from CMINIT	
buffer	Input	Name of buffer area containing data to be sent	
send_length	Input	Length of data in buffer to be sent	0 <= data < 32K

Parameter	In/Out	Value	Usage
req_to_send_received	Output	The name of the variable that contains the indication that indicates CM_REQ_TO_SEND_ RECEIVED	Partner request to change state to send
		CM_REQ_TO_SEND_ NOT_RECEIVED	Partner has not requested a state change
return_code	Output	See list of CPI-C return codes	If not CM_OK, the conversation is deallocated

Parameter	In/Out	Value	Usage
conversation _ID	Output	Returned from the LU	
return_code	Output	See list of CPI-C return codes	If not CM_OK, the accept fails and no conversation is started

3 Sample VTAM Configurations

This section provides sample environments showing how BEA eLink Adapter for Mainframe (eAM) software can be configured for use with an Ethernet LAN and an APPN system 390. Considerations for token ring and subarea-style configurations are included. It is assumed that hardware and operating system installation have been completed.

This section discusses the following topics:

Overview of the eAM Environment

An environment properly configured for the BEA eLink Adapter for Mainframe system involves two general components, a local environment and a remote environment.

Local Environment

A local environment is a UNIX-based machine running eAM software. BEA eLink Adapter for Mainframe is a fully bi-directional program, supporting the local system as either a client or server. This environment consists of the following components:

- Hardware consisting of any workstation and network interface supported by the required software.
- Platform operating systems with protocol stacks (PU servers)

Remote Environment

A remote environment is an IBM mainframe that may or may not be on the same local network. As in the local environment, BEA eLink Adapter for Mainframe is a fully bi-directional program, supporting the remote system as either a client or server. This environment consists of the following components:

- Hardware consisting of any workstation and network interface supported by the required software.
- Software as described in "Supported Host Platforms," in the *BEA eLink Adapter for Mainframe Release Notes*.

Sample Environments

Samples of each environment are provided to illustrate a starting point when first configuring your system. These samples are not intended to be used without modifications. Any similarity between them and any actual system is coincidental.

Machine Attributes (LAN Descriptions)

The attributes of the sample environment machines are listed below for reference. The sample configurations refer to these attributes as required.

10BaseT carrying SNA/DLC (IEEE 802.3) and TCP/IP (DIX) traffic.

Name	Attribute
OS	Solaris 2.5.1 or 2.6
SNA	Brixton/CNT 4.1
APP	Tuxedo 6.5
MAC	08:00:20:7C:47:50
IP	206.189.43.14
NAME	beasun2

Name	Attribute
OS	Solaris 7 or Solaris 8
SNA	SunLink 9.1
APP	Tuxedo 6.5
MAC	08:00:20:87:47:2d
IP	206.189.43.54
NAME	dalsun4

Name	Attribute
OS	HP-US B 11.00
SNA	HP SNAPlus 6.0
APP	Tuxedo 6.5
MAC	08:00:09:30:24:77
IP	206.189.43.13

Name	Attribute
NAME	dalhp10

Name	Attribute
OS	MVS 5.22 9510
SNA	VTAM 4.3
APP	CICS 4.1 / IMS DC 5.1
MAC	10:00:5a:d4:3e:8e
IP	206.189.43.98
NAME	beavs

Name	Attribute
OS	OS/390 1.2
SNA	VTAM 4.3
APP	CICS 4.1
MAC	10:00:5a:d4:c1:e0
IP	206.189.43.96
NAME	dalvs2

Name	Attribute
OS	Windows NT Server 4.0, SP2
SNA	IBM Comm Server 6.0
APP	CICS 4.1
MAC	10:00:5a:d4:c1:e0
IP	206.189.43.99
NAME	dalnt

Local Environment Configuration

HPSNAPlus2 configurations are usually set up using the HP xSnapAdmin utility, resulting in the configuration text file `/etc/opt/sna/sna_node.cfg`. This file can be manually created and/or maintained using a text editor, however, using the HP xSnapAdmin utility is recommended. The example below is the `sna_node.cfg` file for the sample environment.

HPSNAPlus2 Configuration

```
[define_node_config_file]
major_version = 5
minor_version = 1
update_release = 1
revision_level = 116

[define_node]
node_name = dalhp10
description = snacrm development
node_type = END_NODE
fqcp_name = BEALAN.DALHP10
cp_alias = dalhp10
mode_to_cos_map_supp = NO
mds_supported = YES
node_id = <05ffffff>
max_locates = 100
dir_cache_size = 255
```

```
max_dir_entries = 0
locate_timeout = 60
reg_with_nn = YES
reg_with_cds = YES
mds_send_alert_q_size = 100
cos_cache_size = 24
tree_cache_size = 40
tree_cache_use_limit = 40
max_tdm_nodes = 0
max_tdm_tgs = 0
max_isr_sessions = 1000
isr_sessions_upper_threshold = 900
isr_sessions_lower_threshold = 800
isr_max_ru_size = 16384
isr_rcv_pac_window = 8
store_endpt_rscvs = NO
store_isr_rscvs = NO
store_dlur_rscvs = NO
dlur_support = YES
pu_conc_support = NO
nn_rar = 128
ptf_flags = NONE

[define_ethernet_dlc]
dlc_name = ETHER0
description = ""
neg_ls_supp = YES
initially_active = NO
adapter_number = 0

[define_ethernet_port]
port_name = ethl0
description = 1st ethernet adapter
dlc_name = ETHER0
port_type = PORT_SATF
port_number = 1
max_rcv_btu_size = 1033
tot_link_act_lim = 64
inb_link_act_lim = 0
out_link_act_lim = 0
ls_role = LS_NEG
act_xid_exchange_limit = 9
nonact_xid_exchange_limit = 5
ls_xmit_rcv_cap = LS_TWS
max_ifrm_rcvd = 7
target_pacing_count = 7
max_send_btu_size = 1033
mac_address = <000000000000>
lsap_address = 0x08
```

```
implicit_cp_cp_sess_support = NO
implicit_limited_resource = NO
implicit_deact_timer = 0
effect_cap = 3993600
connect_cost = 0
byte_cost = 0
security = SEC_NONSECURE
prop_delay = PROP_DELAY_LAN
user_def_parm_1 = 0
user_def_parm_2 = 0
user_def_parm_3 = 0
initially_active = YES
test_timeout = 5
test_retry_limit = 2
xid_timeout = 5
xid_retry_limit = 2
t1_timeout = 5
t1_retry_limit = 5

[define_ethernet_ls]
ls_name = P390HP10
description = P390 – beavs
port_name = ethl0
adj_cp_name = P390.USS3270
adj_cp_type = LEARN_NODE
mac_address = <0020af543176>
lsap_address = 0x08
auto_act_supp = NO
tg_number = 0
limited_resource = NO
solicit_sscp_sessions = NO
pu_name = <0000000000000000>
disable_remote_act = NO
default_nn_server = NO
dspu_services = NONE
dspu_name = <0000000000000000>
dlus_name = <00000000000000000000000000000000>
bkup_dlus_name = <00000000000000000000000000000000>
link_deact_timer = 0
use_default_tg_chars = YES
ls_attributes = SNA
adj_node_id = <00000000>
local_node_id = <00000000>
cp_cp_sess_support = YES
effect_cap = 3993600
connect_cost = 0
byte_cost = 0
security = SEC_NONSECURE
prop_delay = PROP_DELAY_LAN
```

```
user_def_parm_1 = 0
user_def_parm_2 = 0
user_def_parm_3 = 0
target_pacing_count = 7
max_send_btu_size = 1033
ls_role = USE_PORT_DEFAULTS
initially_active = NO
react_timer = 30
react_timer_retry = 65535
test_timeout = 5
test_retry_limit = 2
xid_timeout = 5
xid_retry_limit = 2
t1_timeout = 5
t1_retry_limit = 5

[define_local_lu]
lu_name = LUHP10A
description = Test LU #1
lu_alias = LUHP10A
nau_address = 0
syncpt_support = YES
lu_session_limit = 0
default_pool = NO
pu_name = <0000000000000000>
sys_name = ""
timeout = -1
back_level = NO

[define_local_lu]
lu_name = LUHP10B
description = Test LU #2
lu_alias = LUHP10B
nau_address = 0
syncpt_support = YES
lu_session_limit = 0
default_pool = NO
pu_name = <0000000000000000>
sys_name = ""
timeout = -1
back_level = NO

[define_local_lu]
lu_name = LUHP10C
description = Test LU #3
lu_alias = LUHP10C
nau_address = 0
syncpt_support = YES
lu_session_limit = 0
```

```
default_pool = NO
pu_name = <0000000000000000>
sys_name = ""
timeout = -1
back_level = NO

[define_partner_lu]
description = APPC MVS LU for IMS
fqplu_name = P390.MVSLU01
plu_alias = MVSLU01
plu_un_name = MVSLU01
max_mc_ll_send_size = 32767
conv_security_ver = NO
parallel_sess_supp = YES

[define_partner_lu]
description = backend cics #1
fqplu_name = P390.C410XB01
plu_alias = C410XB01
plu_un_name = C410XB01
max_mc_ll_send_size = 32767
conv_security_ver = NO
parallel_sess_supp = YES

[define_partner_lu]
description = Second backend cics
fqplu_name = P390.C410XB02
plu_alias = CICS2
plu_un_name = C410XB02
max_mc_ll_send_size = 32767
conv_security_ver = NO
parallel_sess_supp = YES

[define_mode]
mode_name = SMSNA100
description = Sessions: 10 -- 5,5
max_ru_size_upp = 1024
receive_pacing_win = 4
default_ru_size = YES
max_neg_sess_lim = 256
plu_mode_session_limit = 10
min_conwin_src = 5
cos_name = #eLink
cryptography = NONE
auto_act = 0
```

Microsoft SNA Cross-Platform Definitions

Be sure to communicate with the administrator of the CICS/ESA remote domain to obtain key parameters in the VTAM definition that must be included in the Microsoft SNA Server configuration, as well as in other configuration files in the eAM local domain.

Before installing eAM software, please examine the following general procedure for configuring the Microsoft SNA Server. Use the Microsoft SNA Server Manager GUI. Sample values are shown in parenthesis. Consult with the VTAM system administrator to obtain the proper values.

1. Start Microsoft SNA Server Manager from Start button on the Task Bar.
2. A server is automatically created (MVSNT1). Note the configuration values displayed in the Server Properties window:

   Server: MVSNT1

   Subdomain: MVSNT1

   Server Role: Primary

   Network Transports: TCP/IP

3. Under Link Services, define a link service (SNADLC1)

   In the link service Properties, define DLC 802.2 Link Service Configuration:

   Title: DLC 802.2 Link Service #1

   Adapter: <your ethernet adapter>

   Local Service Access Point (SAP): 0x4

   Use Fixed SAP

4. Under SNA Service, Connections, define an 802.2 connection (MVSNT1.).

   In the MVSNT1 Properties, define:

   General

   Name: MVSNT1

   Link Service: SnaDlc1

   Remote End: Peer System

   Allowed Directions: Both Directions

Activation: On Server Startup

Supports Dynamic Remote APPC LU Definition

Address

Remote Network Address: <host MAC address>

Remote SAP Address: <host SAP address>

System Identification

Local Node Name

Network Name: <mynetwork>

Control Point Name: MVSNT1

Local Node ID: <xxx nnnn>XID Type: Format 3

Remote Node Name

Network Name: <hostnetwork>

Control Point Name: <vtamcpname>

Remote Node ID: Peer DLC Role: Negotiable

Compression Type: None

802.2 DLC

Take Defaults

5. Under Local APPC LUs (SNA Service: Connections: Insert: APPC: Local LU), define a local lu (LUNT1A) in the LUNT1A Properties, define:

   General

   LU Alias: LUNT1A

   Network Name: <mynetwork>

   LU Name: LUNT1A

   Advanced

   Take Defaults

6. Under Remote APPC LUs, define a remote lu (CICS1) in the CICS1 Properties, define:

   General

Connection: MVSNT1

LU Alias: CICS1

Network Name: <hostnetwork>

LU Name: CICS1

Uninterpreted Name: CICS1

Options

Take Defaults

7. Under APPC Modes, define a mode (SMSNA100) in the SMSNA100 Properties, define:

   General

   Mode Name: SMSNA100

   Limits

   Parallel Session Limit: <max sessions>

   Minimum Winner Contention Limit: <min winners>

   Partner Min Winner Contention Limit: <max sessions - min winners>

   Automatic Activation Limit: 0

   Characteristics

   Take Defaults

   Partners

   Add partnership for Server Name: MVSNT1 between Local LU: LUNT1A and Partner LU: CICS1

   Compression

   Take Defaults

VTAM Application Program Major Node

The APPLID definition shown in the following listing defines the local stack configuration to run under OS/390 using VTAM.

Listing 3-1 Applid definition (OS/390)

```
BEAVTAM VBUILD TYPE=APPL
BEAAPPL1 APPL ACBNAME=BEAAPPL1,
              AUTH=(ACQ,PASS),
              APPC=YES,
              SYNCLVL=SYNCPT,
              PARSESS=YES
```

Remote Environment Configurations

You must involve your mainframe system support personnel early in the process. Make sure everyone is involved. Most of the configuration for your mainframe should have already been done.

These samples are provided for illustration. Mainframe technical support is important. This documentation is not intended demonstate all of the possible configurations. These samples represent one way a P390 can be configured to work in an APPN LAN environment.

ATCSTRxx VTAM Start List

The example below is the VTAM start list for the BEAVS P390 machine. It supports both the subarea and APPN environments.

```
* -------------------------*
* VTAM START LIST FOR SYS1  *
* -------------------------*
BN=YES,
BNDYN=FULL,
XNETALS=YES,
SSCPID=06,NOPROMPT,
CONFIG=00,MAXSUBA=31,SUPP=NOSUP,
SSCPNAME=USS3270,
NETID=P390,
NODETYPE=NN,
HOSTSA=6,
CRPLBUF=(208,,15,,1,16),
IOBUF=(100,512,19,,1,20),
LFBUF=(104,,0,,1,1),
```

```
LPBUF=(64,,0,,1,1),
SFBUF=(163,,0,,1,1)
```

XCA Major Node Defines the LAN Adapter for SYS1

This definition is set up for use with a 3172 (emulated) for connecting an APPN network node to another APPN node. Note that it is for an Ethernet LAN and the SAPADDR specified must be the same as the LSap specified for the local link station.

```
*****************************************************
*  Emulated 3172 XCA MAJOR NODE FOR HOST beavs
*****************************************************
XETH2LP1 VBUILD TYPE=XCA ** EXTERNAL COMMUNICATION ADAPT**
PORTE2   PORT  ADAPNO=1,       ** 3172 RELATIVE ADAPTER NUMBER**
               CUADDR=E22,     ** CHANNEL UNIT ADDRESS        **
               MEDIUM=CSMACD,  ** LAN TYPE=ETHERNET           **
               SAPADDR=8,      ** SERVICE ACCESS POINT ADDRESS**
               TIMER=120       ** CHANNEL ACTIVATE RESP TIME  **
*
G1ETH2   GROUP DIAL=YES,       ** YES required for putype 2   **
               DYNPU=YES,
               CALL=INOUT,
               ANSWER=ON,
               ISTATUS=ACTIVE
LETH20   LINE
PETH20   PU
LETHE3   LINE
PETHE3   PU
LETHF3   LINE
PETHF3   PU
```

Switched Network (SWNET) Definitions

The three switched network definition examples in this section specify the VTAM PU, representing the local link stations that expect to connect with the host machine. The 10BLK and IDNUM definitions are provided to support 3270 traffic and must be unique, as well as match the values specified in the local link definition.

SWNET Major Node (DALHP10)

```
SWNETHHP   VBUILD  TYPE=SWNET,MAXNO=3,MAXGRP=3
P390HP10   PU  ADDR=02,
               IDBLK=05F,
               IDNUM=FFFFF,
               PUTYPE=2,
```

```
                NETID=BEALAN,
                CPNAME=DALHP10,
                MAXPATH=3,
                DWACT=YES,
                CONNTYPE=APPN,
                CPCP=YES,
                DYNLU=YES
* ------------------------------------------------
* SNA SAP & HP10 MAC ADDRESS BIT REVERSED FOR TRFMT
* ------------------------------------------------
PATHHP      PATH DIALNO=00041000900C24EE,
                GRPNM=G1ETH2
LUHP10A   LU  LOCADDR=0
LUHP10B   LU  LOCADDR=0
LUHP10C   LU  LOCADDR=0
```

SWNET Major Node (SUN2)

```
SWNETH2     VBUILD  TYPE=SWNET,MAXNO=3,MAXGRP=3
P390ETH2    PU  ADDR=04,
                IDBLK=019,
                IDNUM=10092,
                PUTYPE=2,
                NETID=BEALAN,
                CPNAME=SUN2,
                MAXPATH=3,
                DWACT=YES,
                CONNTYPE=APPN,
                CPCP=YES,
                DYNLU=YES
* ------------------------------------------------
* SNA SAP & SUN2 MAC ADDRESS BIT REVERSED FOR TRFMT
* ------------------------------------------------
PATH01      PATH DIALNO=00081000043EE20A,
                GRPNM=G1ETH2
LUSUN2A   LU  LOCADDR=0
LUSUN2B   LU  LOCADDR=0
LUSUN2C   LU  LOCADDR=0
```

SWNET Major Node ((SUN4)

```
SWNESUN4    VBUILD  TYPE=SWNET,MAXNO=3,MAXGRP=3
P390ETH4    PU  ADDR=03,
                IDBLK=018,
                IDNUM=10092,
                PUTYPE=2,
                NETID=BEALAN,
```

```
                CPNAME=SUN4,
                MAXPATH=3,
                DWACT=YES,
                CONNTYPE=APPN,
                CPCP=YES,
                DYNLU=YES
* ------------------------------------------------
* SNA SAP & SUN4 MAC ADDRESS BIT REVERSED FOR TRFMT
* ------------------------------------------------
PATHSL4    PATH DIALNO=0008100002E1E2B4,
                GRPNM=G1ETH2
LUSUN4A  LU  LOCADDR=0
LUSUN4B  LU  LOCADDR=0
LUSUN4C  LU  LOCADDR=0
```

VTAM Application Major Nodes for CICS Regions

These examples represent the partner LU definitions to be accessed from the local environment. The APPL names must match those specified in the partner LU definitions on the local machine.

```
BEACICS VBUILD TYPE=APPL                APPLICATION MAJOR NODE
* APPL DEFINITION STATEMENTS FOR CICS
* CICS 4.10 BACKEND REGION #1 SYSID=B41A
C410XB01 APPL EAS=64,                ESTIMATED CONCURRENT SESSIONS
              MODETAB=ISTINCLM,      MAKE SURE DEFAULT MODETAB
              PARSESS=YES,
              AUTH=(ACQ,BLOCK,PASS) CICS CAN ACQUIRE & PASS TMLS
                                     CICS CAN REQUEST BLOCKED INPUT

C410XB02 APPL EAS=64,                ESTIMATED CONCURRENT SESSIONS
              MODETAB=ISTINCLM,      MAKE SURE DEFAULT MODETAB
              PARSESS=YES,
              AUTH=(ACQ,BLOCK,PASS) CICS CAN ACQUIRE & PASS TMLS
                                     CICS CAN REQUEST BLOCKED INPUT
C410XB03 APPL EAS=64,                ESTIMATED CONCURRENT SESSIONS
              MODETAB=ISTINCLM,      MAKE SURE DEFAULT MODETAB
              PARSESS=YES,
              AUTH=(ACQ,BLOCK,PASS) CICS CAN ACQUIRE & PASS TMLS
                                     CICS CAN REQUEST BLOCKED INPUT
* ########################
* END OF BEACICS APPL DEF
* ########################
```

CICS Resource Definition Entries (RDO)

CICS connection and session definitions map the VTAM path definitions for the CICS application. Each connection represents one local LU definition in the local SNA configuration, therefore, the names must match.

CICS session definitions associate a VTAM mode with the LU specified in the connection. The mode names and session count characteristics must match those specified in the mode definitions for the local SNA configuration. Note that these definitions set AUTOCONNECT to "YES," allowing automatic session acquisition for a CICS client application.

```
LIST GROUP(BEAHP10) OBJECTS
GROUP NAME: BEAHP10
 ----------
 CONNECTIONS:          FHPA     FHPB      FHPC
 SESSION                        FHPA      FHPB   FHPC

CONNECTION(FHPA)       GROUP(BEAHP10)
                       DESCRIPTION(1ST HP SNAP2+ CONNECTION)
  CONNECTION-IDENTIFIERS
     NETNAME(LUHP10A)           INDSYS()
  REMOTE-ATTRIBUTES
     REMOTESYSTEM()  REMOTENAME() REMOTESYSNET()
  CONNECTION-PROPERTIES
     ACCESSMETHOD(VTAM)         PROTOCOL(APPC)    CONNTYPE()
     SINGLESESS(NO)             DATASTREAM(USER)  RECORDFORMAT(U)
     QUEUELIMIT(NO)             MAXQTIME(NO)
  OPERATIONAL-PROPERTIES
     AUTOCONNECT(NO)            INSERVICE(YES)
  SECURITY
     SECURITYNAME()             ATTACHSEC(LOCAL)  BINDSECURITY(NO)
     USEDFLTUSER(NO)
  RECOVERY
     PSRECOVERY(SYSDEFAULT)

CONNECTION(FHPB)       GROUP(BEAHP10)
                       DESCRIPTION(2ND HP SNA+ 2 CONNECTION)
  CONNECTION-IDENTIFIERS
     NETNAME(LUHP10B)           INDSYS()
  REMOTE-ATTRIBUTES
     REMOTESYSTEM()             REMOTENAME()      REMOTESYSNET()
  CONNECTION-PROPERTIES
     ACCESSMETHOD(VTAM)         PROTOCOL(APPC)    CONNTYPE()
     SINGLESESS(NO)             DATASTREAM(USER)  RECORDFORMAT(U)
     QUEUELIMIT(NO)             MAXQTIME(NO)
  OPERATIONAL-PROPERTIES
```

```
      AUTOCONNECT(YES)           INSERVICE(YES)
   SECURITY
      SECURITYNAME()             ATTACHSEC(LOCAL)  BINDSECURITY(NO)
      USEDFLTUSER(NO)
   RECOVERY
      PSRECOVERY(SYSDEFAULT)

CONNECTION(FHPC)      GROUP(BEAHP10)
                      DESCRIPTION(3RD HP SNA+ 2 CONNECTION)
   CONNECTION-IDENTIFIERS
      NETNAME(LUHP10C)           INDSYS()
   REMOTE-ATTRIBUTES
      REMOTESYSTEM()             REMOTENAME()      REMOTESYSNET()
   CONNECTION-PROPERTIES
      ACCESSMETHOD(VTAM)         PROTOCOL(APPC)    CONNTYPE()
      SINGLESESS(NO)             DATASTREAM(USER)  RECORDFORMAT(U)
      QUEUELIMIT(NO)             MAXQTIME(NO)
   OPERATIONAL-PROPERTIES
      AUTOCONNECT(NO)            INSERVICE(YES)
   SECURITY
     SECURITYNAME()              ATTACHSEC(LOCAL)  BINDSECURITY(NO)
     USEDFLTUSER(NO)
   RECOVERY
     PSRECOVERY(SYSDEFAULT)

 SESSIONS(FHPA)       GROUP(BEAHP10)
                      DESCRIPTION(1ST HP SNAP2+ SESSION)
    SESSION-IDENTIFIERS
       CONNECTION(FHPA)          SESSNAME()        NETNAMEQ()
       MODENAME(SMSNA100)
    SESSION-PROPERTIES
       PROTOCOL(APPC)            MAXIMUM(32,16)    RECEIVEPFX()
       RECEIVECOUNT()            SENDPFX()         SENDCOUNT()
        SENDSIZE(4096)           RECEIVESIZE(4096) SESSPRIORITY(0)
    PRESET-SECURITY
       USERID()
    OPERATIONAL-PROPERTIES
       AUTOCONNECT(NO)           BUILDCHAIN(YES)   USERAREALEN(0)
       IOAREALEN(0,0)            RELREQ(NO)        DISCREQ(NO)
       NEPCLASS(0)
    RECOVERY
      RECOVOPTION(SYSDEFAULT)

 SESSIONS(FHPB)       GROUP(BEAHP10)
                      DESCRIPTION(2ND HP SNAP2+ SESSION)
    SESSION-IDENTIFIERS
      CONNECTION(FHPB)           SESSNAME()        NETNAMEQ()
      MODENAME(SMSNA100)
    SESSION-PROPERTIES
```

```
    PROTOCOL(APPC)            MAXIMUM(32,16)    RECEIVEPFX()
    RECEIVECOUNT()            SENDPFX()         SENDCOUNT()
    SENDSIZE(4096)            RECEIVESIZE(4096) SESSPRIORITY(0)
  PRESET-SECURITY
    USERID()
  OPERATIONAL-PROPERTIES
    AUTOCONNECT(YES)          BUILDCHAIN(YES)   USERAREALEN(0)
    IOAREALEN(0,0)            RELREQ(NO)        DISCREQ(NO)
    NEPCLASS(0)
  RECOVERY
    RECOVOPTION(SYSDEFAULT)

SESSIONS(FHPC)       GROUP(BEAHP10)
                     DESCRIPTION(3RD HPSNAP2+ SESSION)
  SESSION-IDENTIFIERS
    CONNECTION(FHPC)          SESSNAME()        NETNAMEQ()
    MODENAME(SMSNA100)
  SESSION-PROPERTIES
    PROTOCOL(APPC)            MAXIMUM(10,5)     RECEIVEPFX()
    RECEIVECOUNT()            SENDPFX()         SENDCOUNT()
    SENDSIZE(4096)            RECEIVESIZE(4096) SESSPRIORITY(0)
  PRESET-SECURITY
    USERID()
  OPERATIONAL-PROPERTIES
    AUTOCONNECT(YES)          BUILDCHAIN(YES)   USERAREALEN(0)
    IOAREALEN(0,0)            RELREQ(NO)        DISCREQ(NO)
    NEPCLASS(0)
  RECOVERY
    RECOVOPTION(SYSDEFAULT)
```

4 Application-to-Application Programming Examples

This section provides the following transaction scenarios for the programming environments supported by eAM:

- Distributed Program Link (DPL) Examples
- Distributed Transaction Processing (DTP) Examples
- CPI-C Programming Examples
- CICS/ESA Mirror Transaction Examples

Caution: The scenarios in this section demonstrate how ATMI calls relate to CICS/ESA programming structures. They are not intended for use in developing application code, or for the replacement of existing application code. The use of any of these examples in actual situations may have unpredictable results.

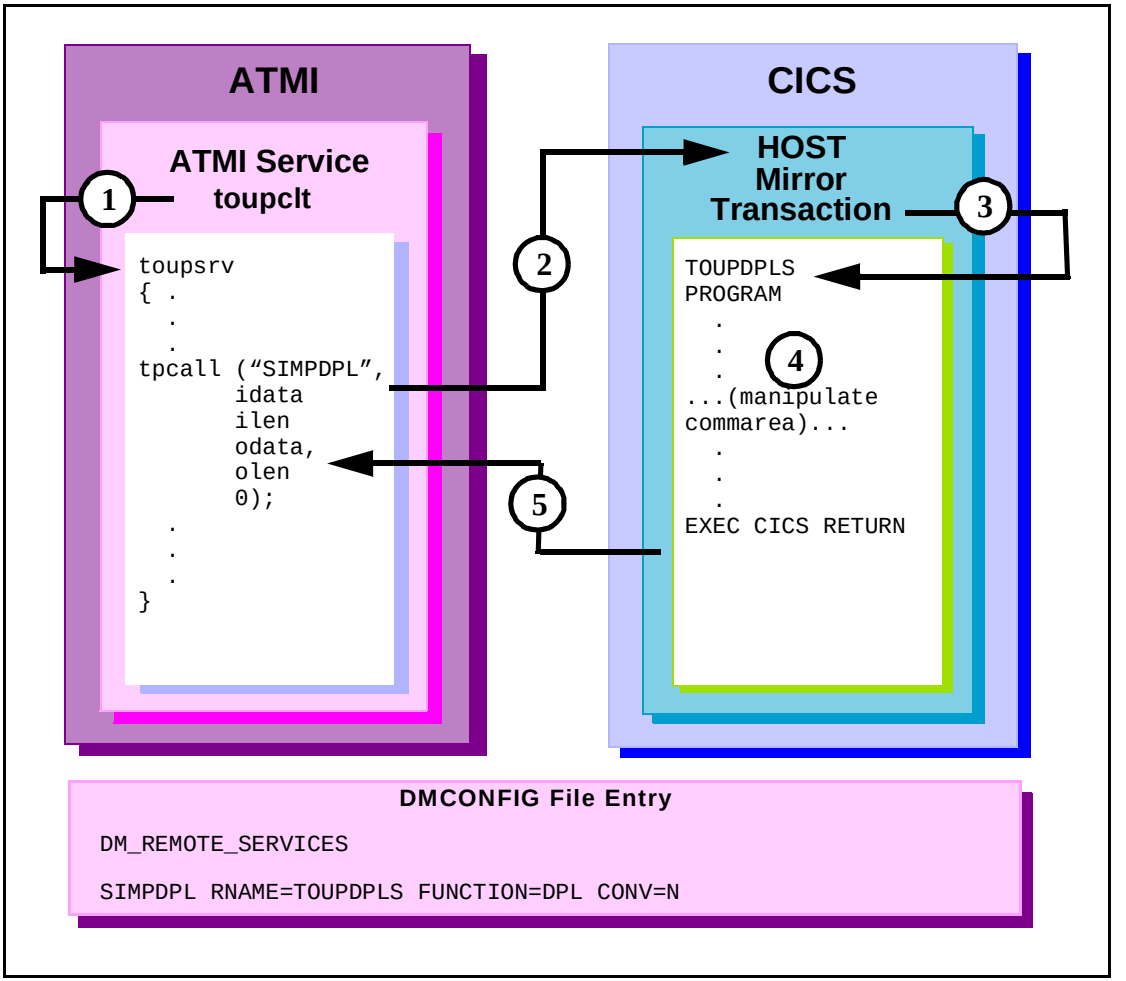
Each example provides a graphical illustration of the scenario followed by a description of each step of the scenario.

Distributed Program Link (DPL) Examples

The examples in this section represent a few of the many programming scenarios available for using DPL and ATMI service invocations. These examples employ the most natural and efficient approaches.

Note: To run transaction client/server scenarios, the eAM software must be licensed for sync-level 2 operations.

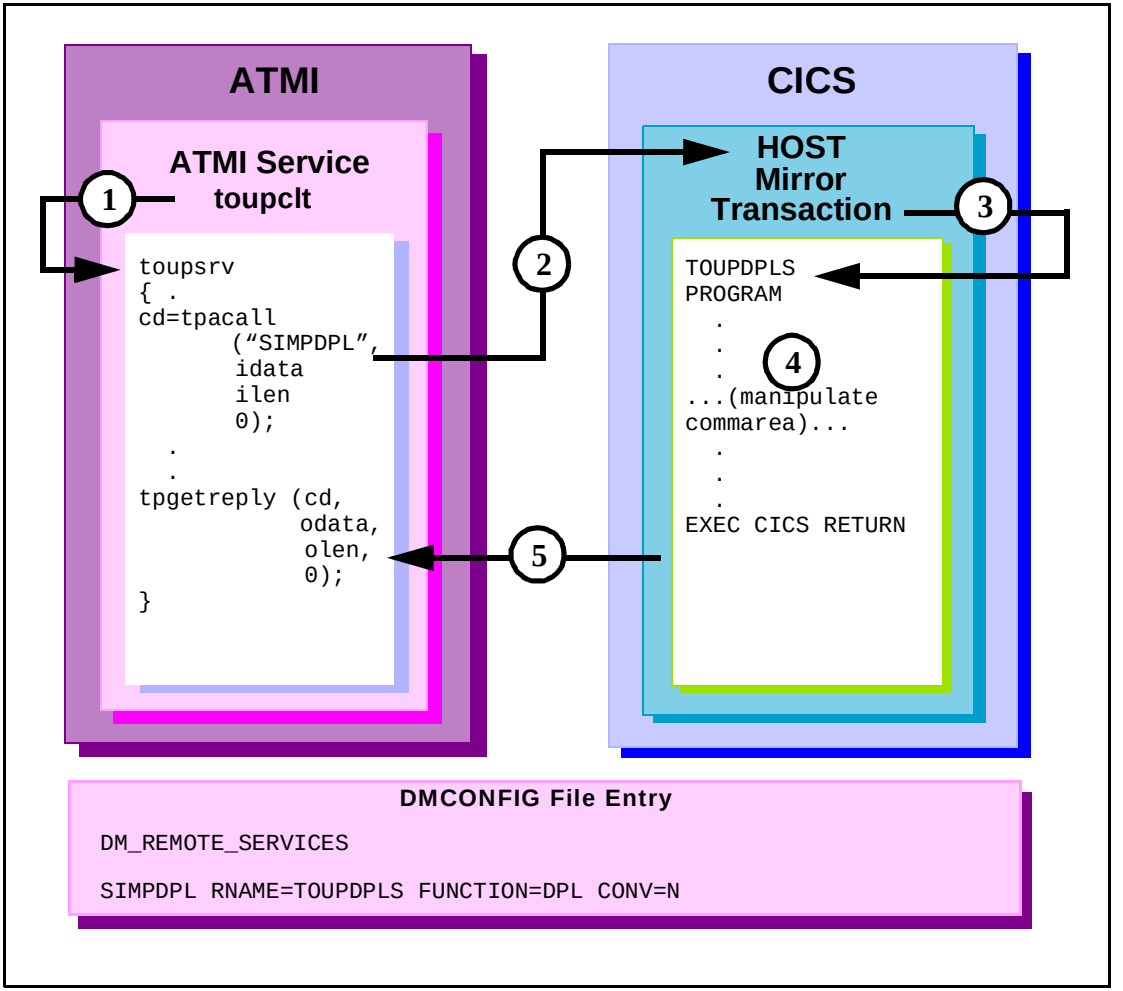
ATMI Client Request/Response to CICS/ESA DPL



1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpcall` for `SIMPDPL`, which is advertised in the `DM_REMOTE_SERVICES` section of the DMCONFIG file.
3. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing.

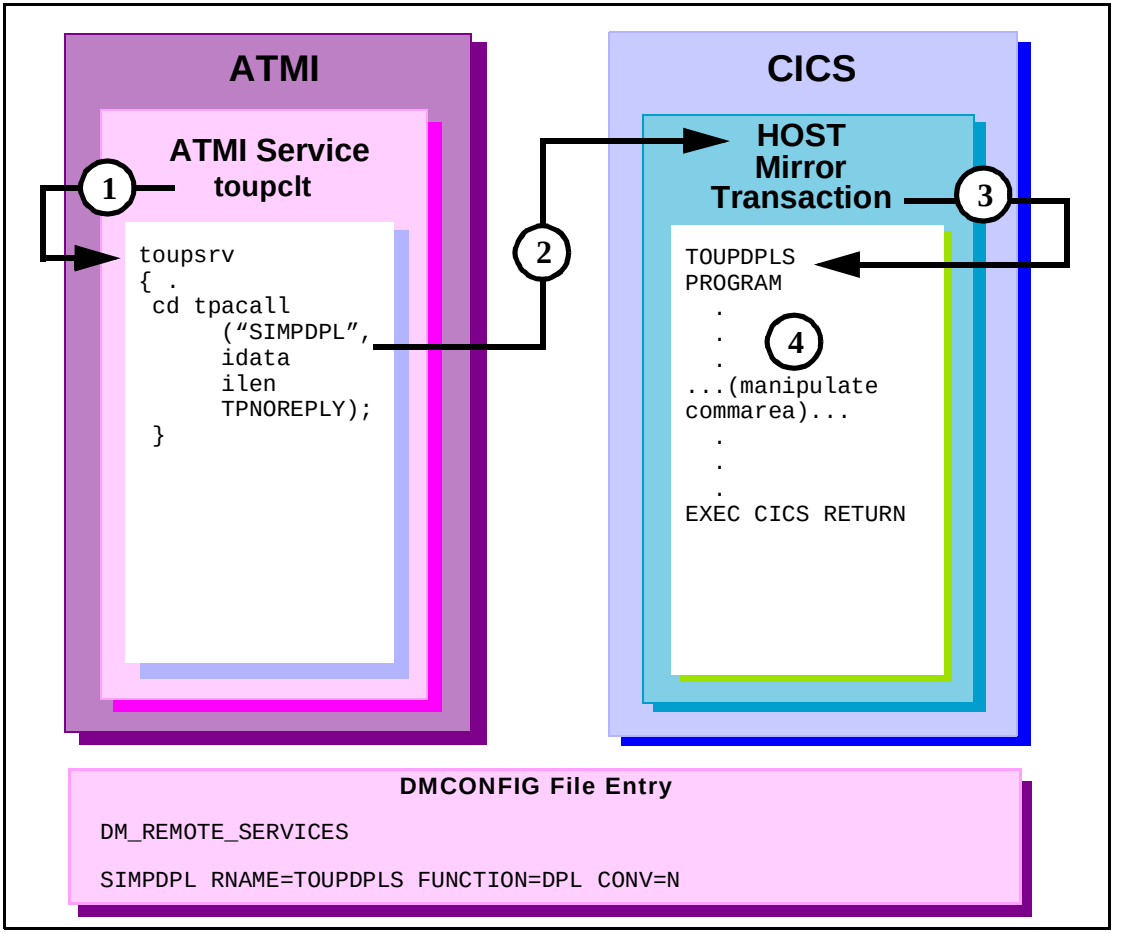
4. The TOUPDPLS program processes data.
5. The CICS/ESA server returns the commarea into the client's odata buffer.

ATMI Client Asynchronous Request/Response to CICS/ESA DPL



1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpacall` for `SIMPDPL`, which is advertised in the `DM_REMOTE_SERVICES` section of DMCONFIG file.
3. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing.
4. The `TOUPDPLS` program processes data.
5. The CICS/ESA system returns the `commarea` into the client's `tpgetreply odata` buffer.

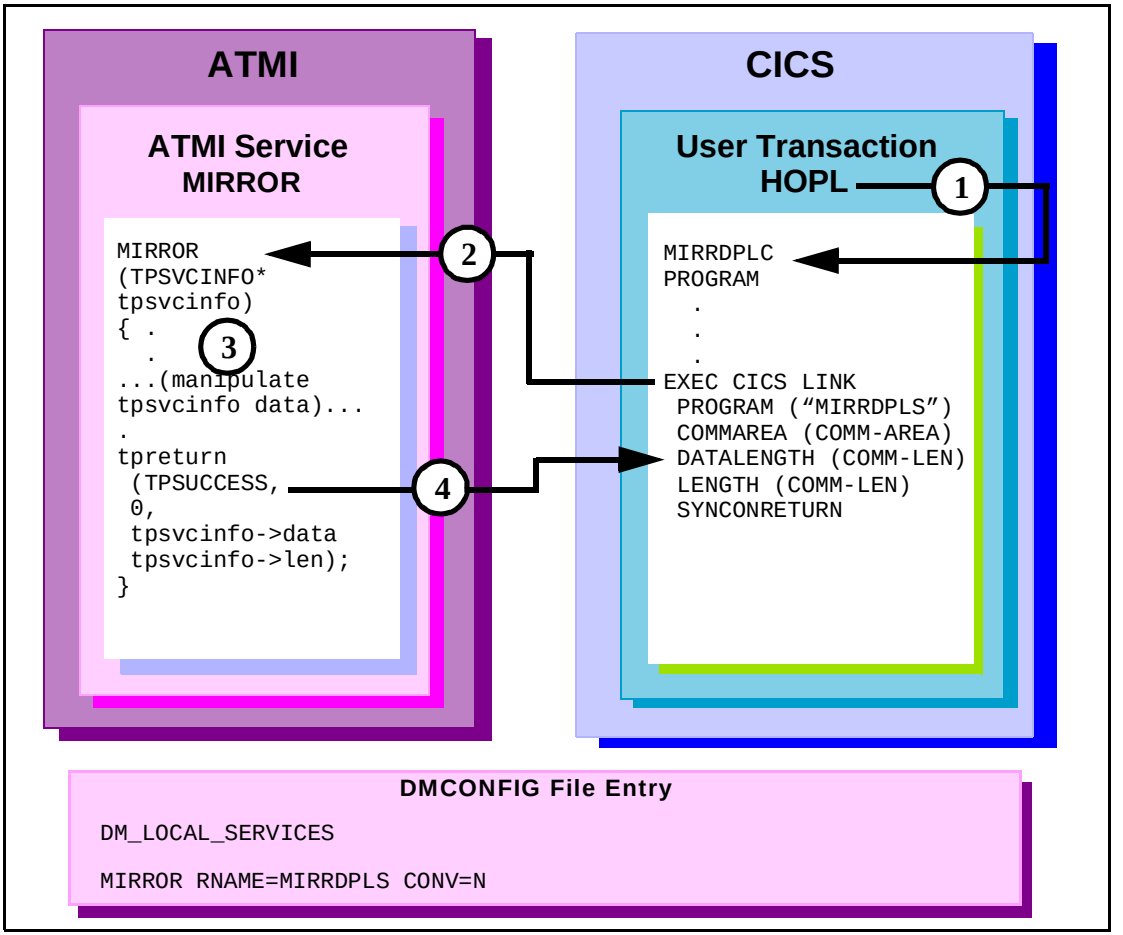
ATMI Client Asynchronous Request/Response with No Reply to CICS/ESA DPL



1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpacall` for `SIMPDPL`, which is advertised in the `DM_REMOTE_SERVICES` section of DMCONFIG file. The `toupsrv` service uses `TPNOREPLY` to specify that no reply is expected.

3. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing.
4. The `TOUPDPLS` program processes data.

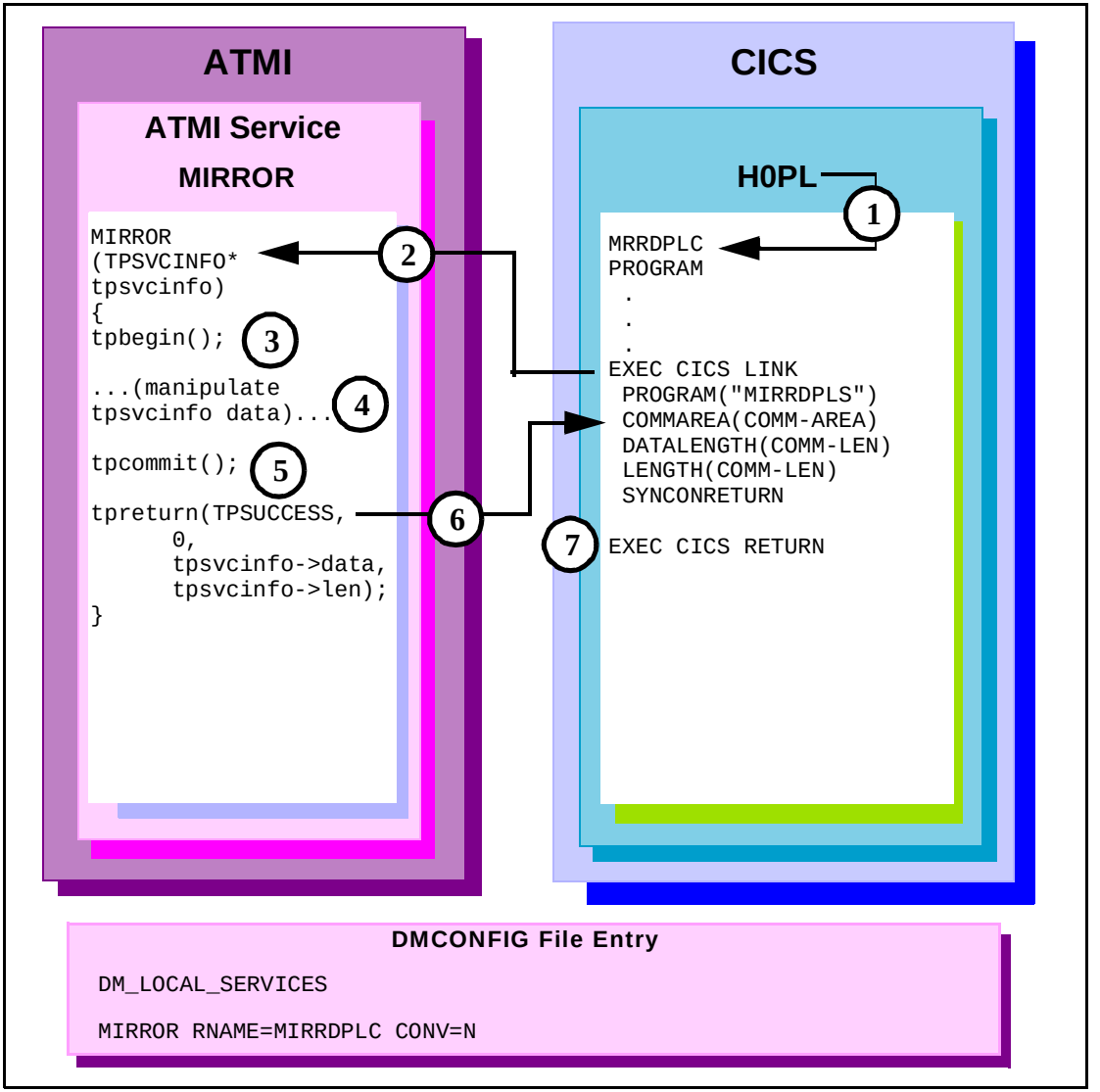
CICS/ESA DPL to ATMI Request/Response Server



1. User-entered `HOPL` invokes `MIRRDPLC` program.
2. The `EXEC CICS LINK` command causes the advertised service mapped to `MIRRDPLS` (in the DM_LOCAL_SERVICES section of the DMCONFIG file) to execute.
3. The `MIRROR` service processes the data received in the service `TPSVCINFO` data buffer from the `EXEC CICS LINK`.

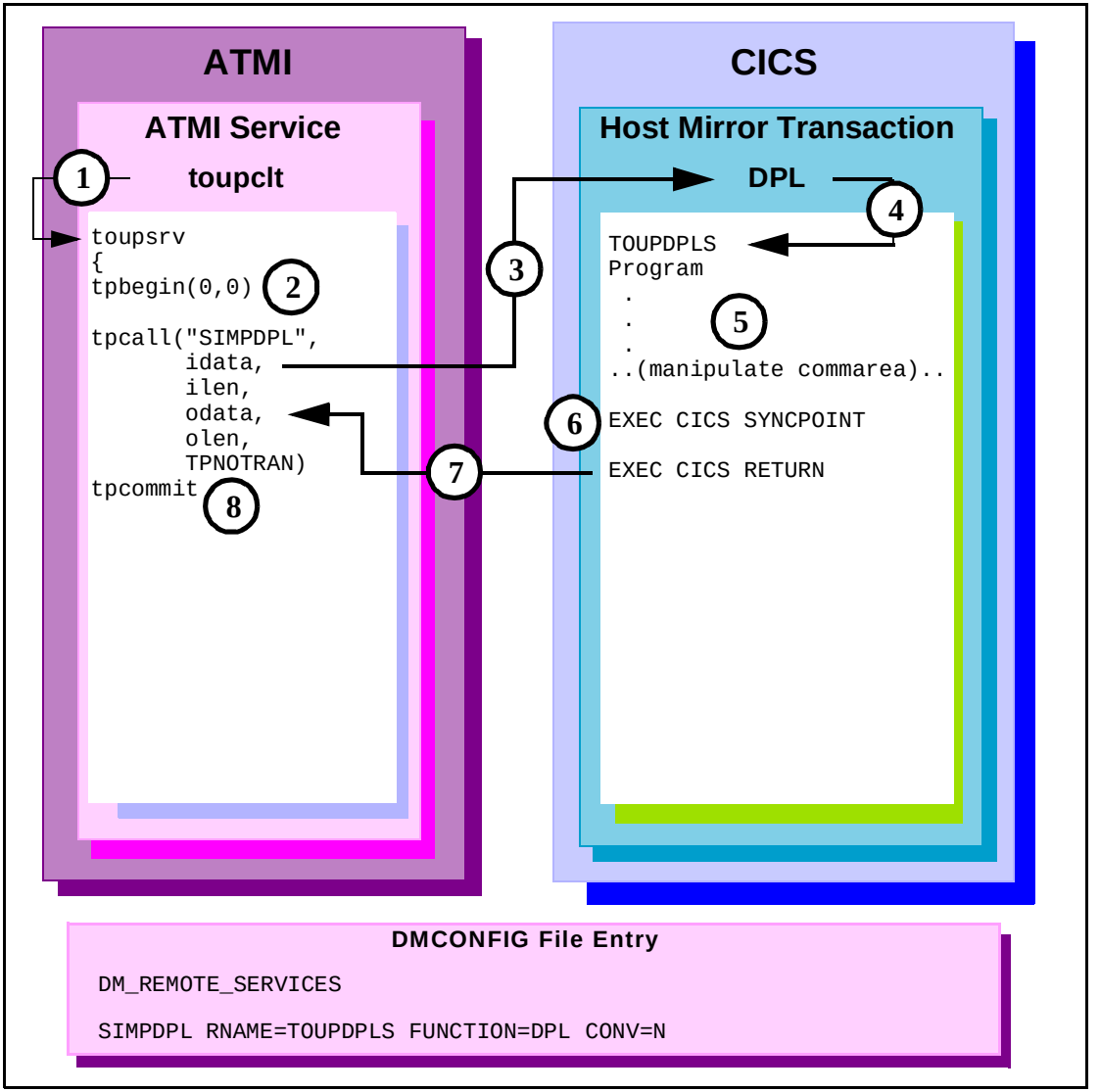
4. The `tpreturn` call returns the data into the `COMM-AREA` buffer.

CICS/ESA DPL to ATMI Request/Response Server, Service in Autonomous Transaction



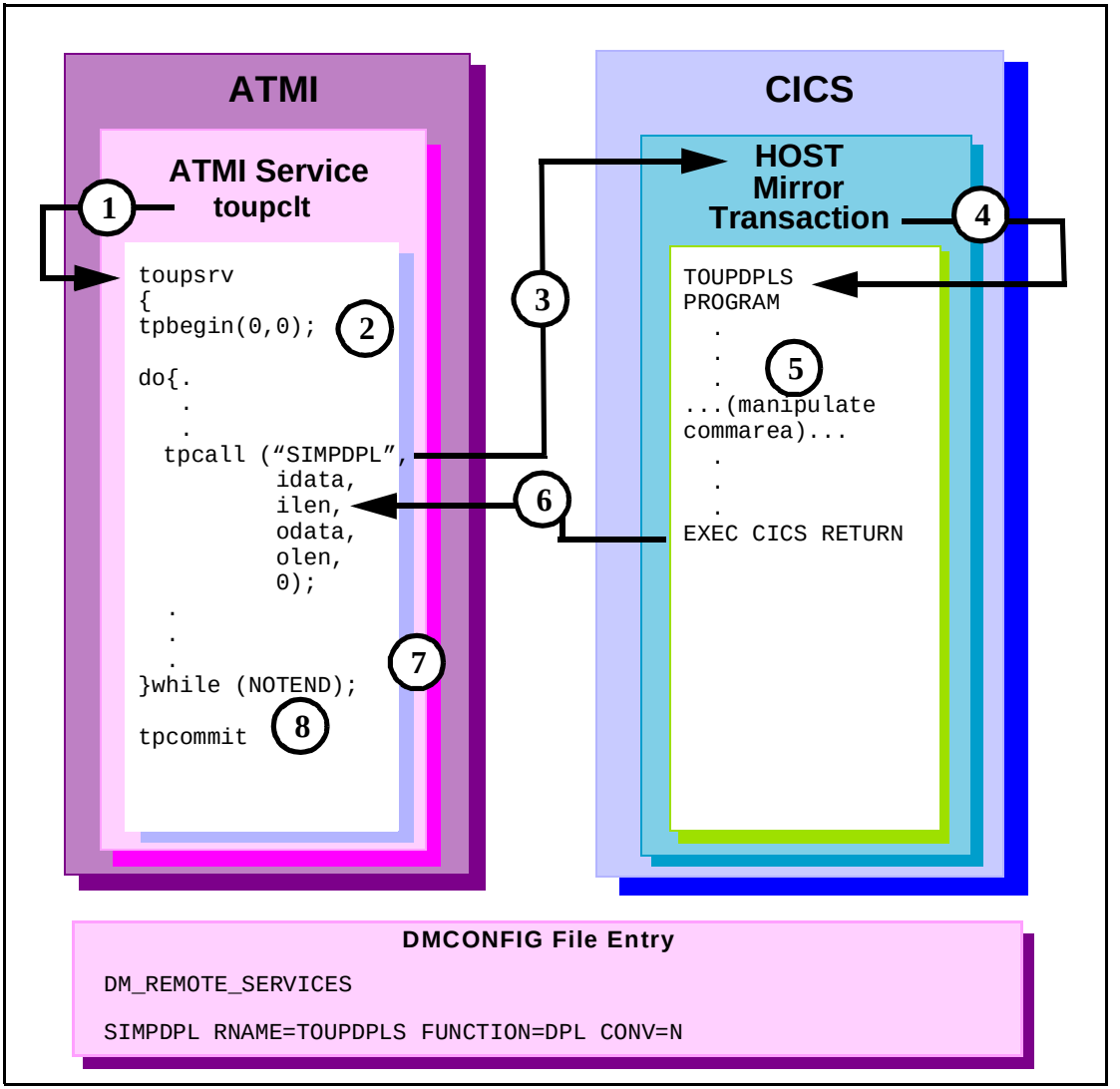
1. User-entered `H0PL` invokes `MIRRDPLC` program.
2. The `EXEC CICS LINK` command causes the advertised service mapped to `MIRRDPLS` (in the DM_LOCAL_SERVICES section of the DMCONFIG file) to execute. The `SYNCONRETURN` option indicates that the invoked service will not participate in the CICS/ESA transaction.
3. The `MIRROR` service request `tpbegin` incorporates all further operations in a transaction.
4. The `MIRROR` service processes the data.
5. The `tpcommit` indicates the end of the transaction; all updates performed within the service transaction are to be committed.
6. The `tpreturn` call returns the data into the `commarea` buffer.
7. The `EXEC CICS SYNCPOINT` is an explicit commit request. All updated resources in the CICS/ESA transaction are committed.

ATMI Client Request/Response to CICS/ESA DPL, in Autonomous Transaction



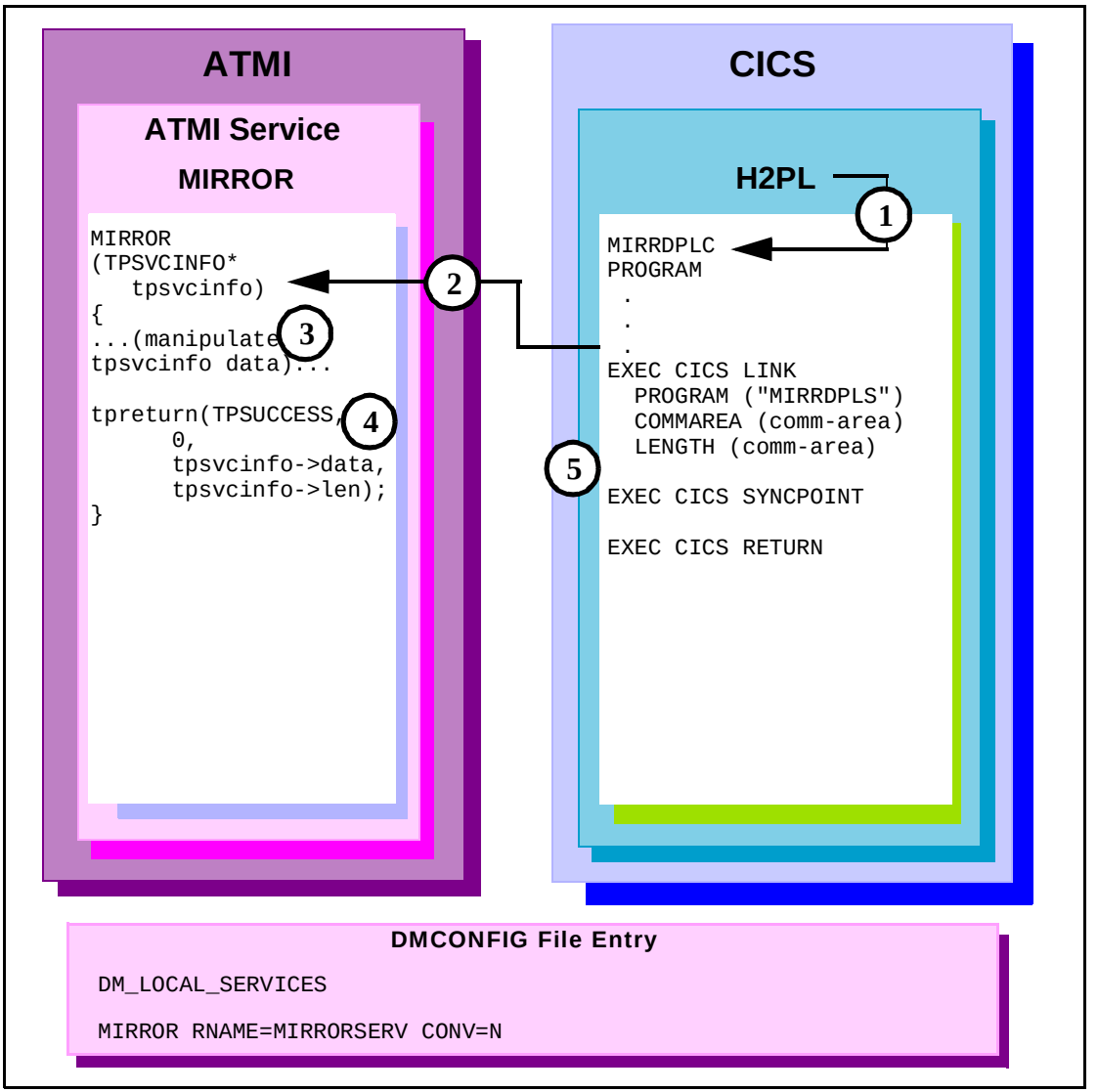
1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.
3. The `toupsrv` service issues `tpcall` for `SIMPDPL`, which is advertised in the DM_REMOTE_SERVICES section of the DMCONFIG file. The `TPNOTRAN` parameter indicates the CICS/ESA application does not participate in the service transaction.
4. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing.
5. The `TOUPDPLS` program processes data.
6. The `EXEC CICS SYNCPOINT` is an explicit commit request. All updated resources in the CICS/ESA transaction are committed.
7. The CICS/ESA server returns the `commarea` into the client's `odata` buffer.
8. The `toupsrv` service `tpcommit` request signals the successful completion of the transaction, causing a commit of its own updated resources.

Transactional ATMI Client Multiple Requests/Responses to CICS/ESA DPL



1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.
3. The `toupsrv` service issues `tpcall` for `SIMPDPL`, which is advertised in the `DM_REMOTE_SERVICES` section of the DMCONFIG file. The `tpcall` is requested multiple times within the same transaction.
4. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing. The host mirror transaction remains as a long-running task to service all further requests on the transaction.
5. The `TOUPDPLS` program processes data.
6. The CICS/ESA system returns the `commarea` into the client's `odata` buffer.
7. Step 3 through Step 6 are repeated until the `toupsrv` service loop end conditions are met.
8. The `tpcommit` request indicates the successful completion of the transaction, causing a commit of its own resources and the resources held by the host mirror transaction.

Transactional CICS/ESA DPL to ATMI Request/Response Server



1. User-entered `H2PL` invokes `MIRRDPLC` program.
2. The `EXEC CICS LINK` command causes the advertised service mapped to `MIRRDPLS` (in the DM_LOCAL_SERVICES section of the DMCONFIG file) to execute. The invoked service participates in the CICS/ESA transaction.
3. The `MIRROR` service processes the data.
4. The `tpreturn` call returns the data into the `commarea` buffer.
5. The `EXEC CICS SYNCPOINT` is an explicit commit request indicating a successful end of the conversation. All updated resources in the transaction are committed.

Distributed Transaction Processing (DTP) Examples

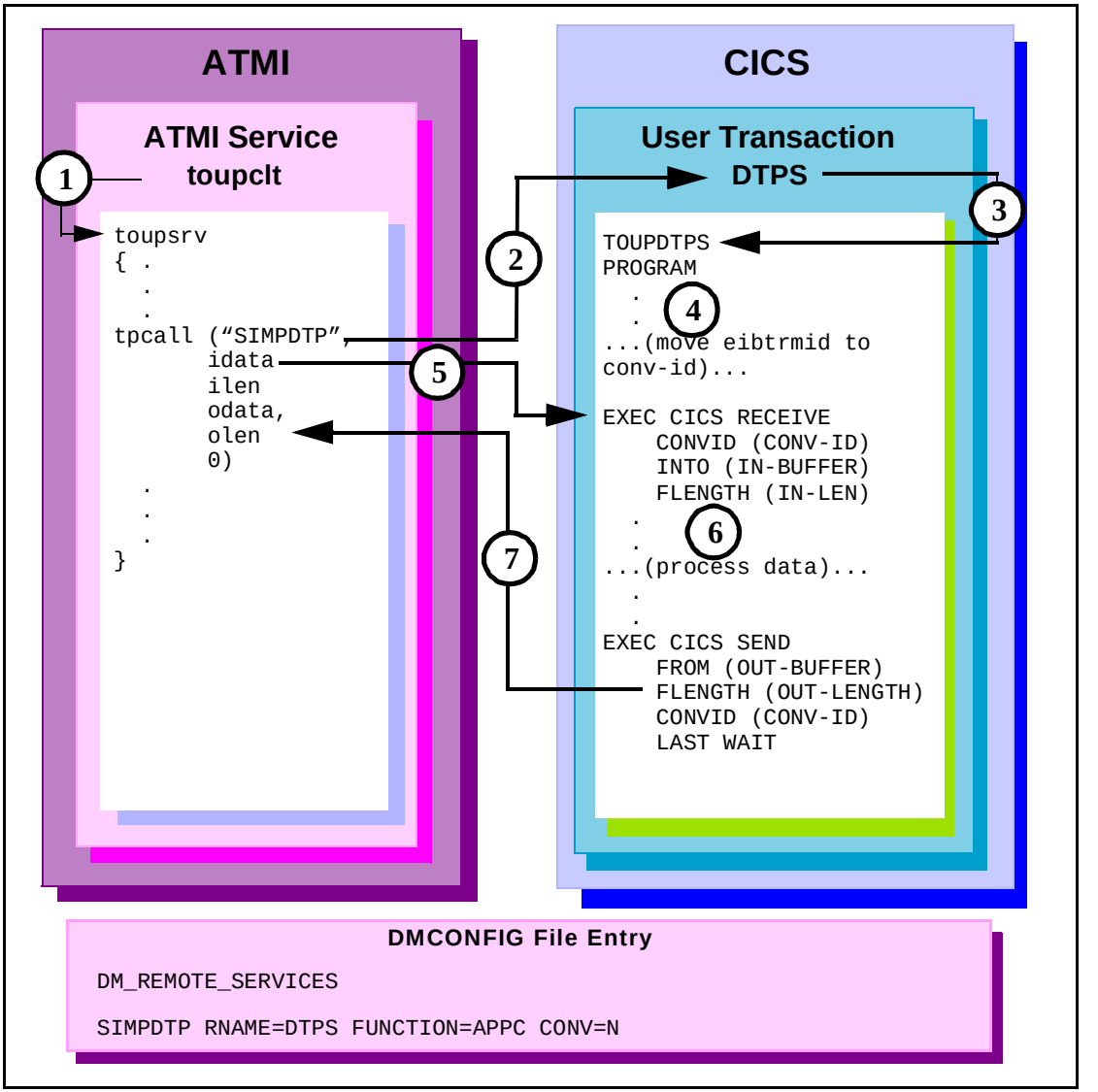
The following examples represent programming scenarios for using DTP and ATMI service invocations.

Although it is most suited for the DPL environment, the `tpcall` is usually used for the DPL environment, it can also be used for a request/response to a DTP server.

The examples in this section represent some of the programming scenarios available for using DTP and ATMI service invocations. These examples employ the most natural and efficient approaches.

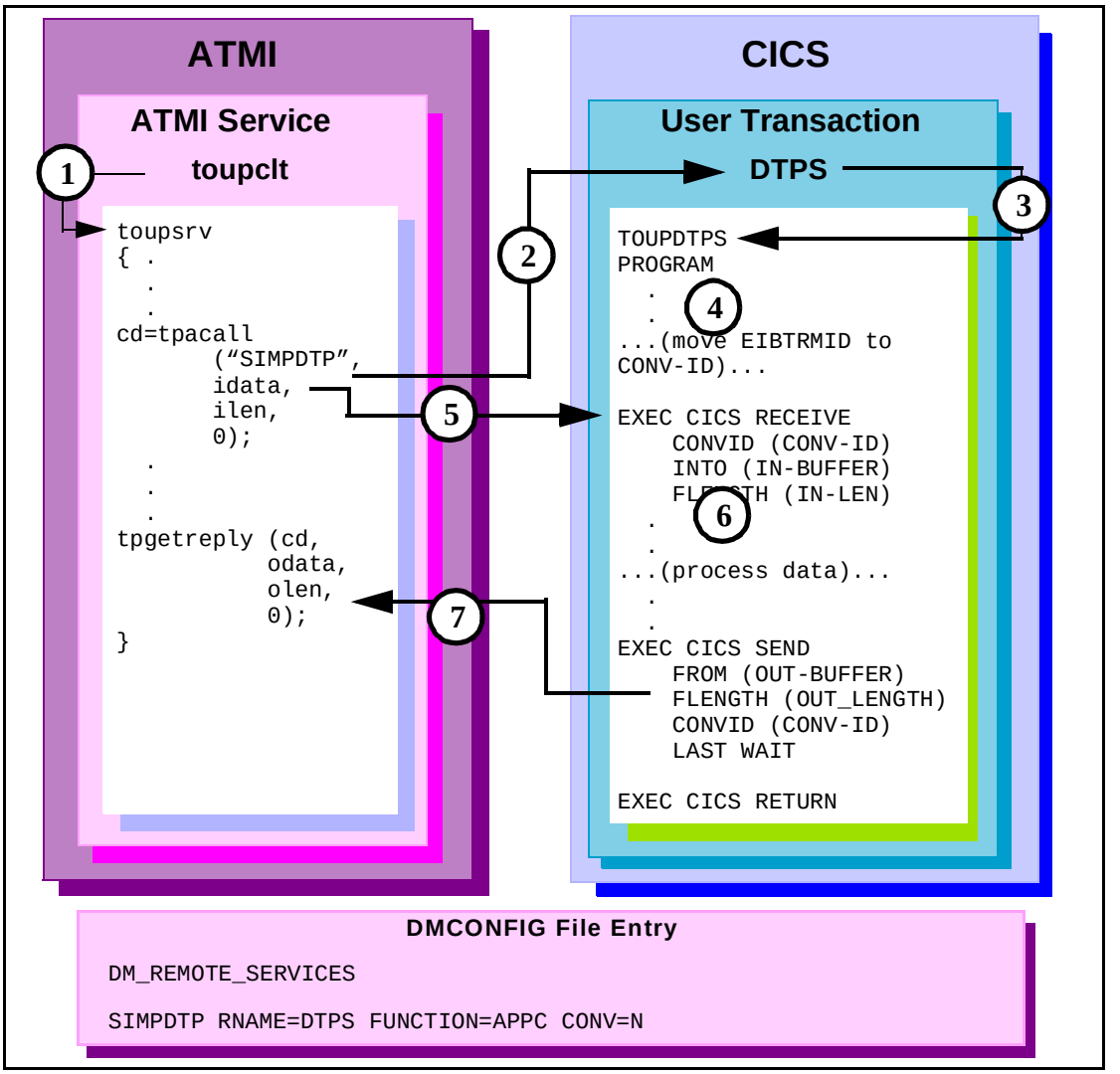
Note: To run transactional client/server scenarios, the eAM software must be licensed for sync-level 2 operations.

ATMI Client Request/Response to CICS/ESA DTP



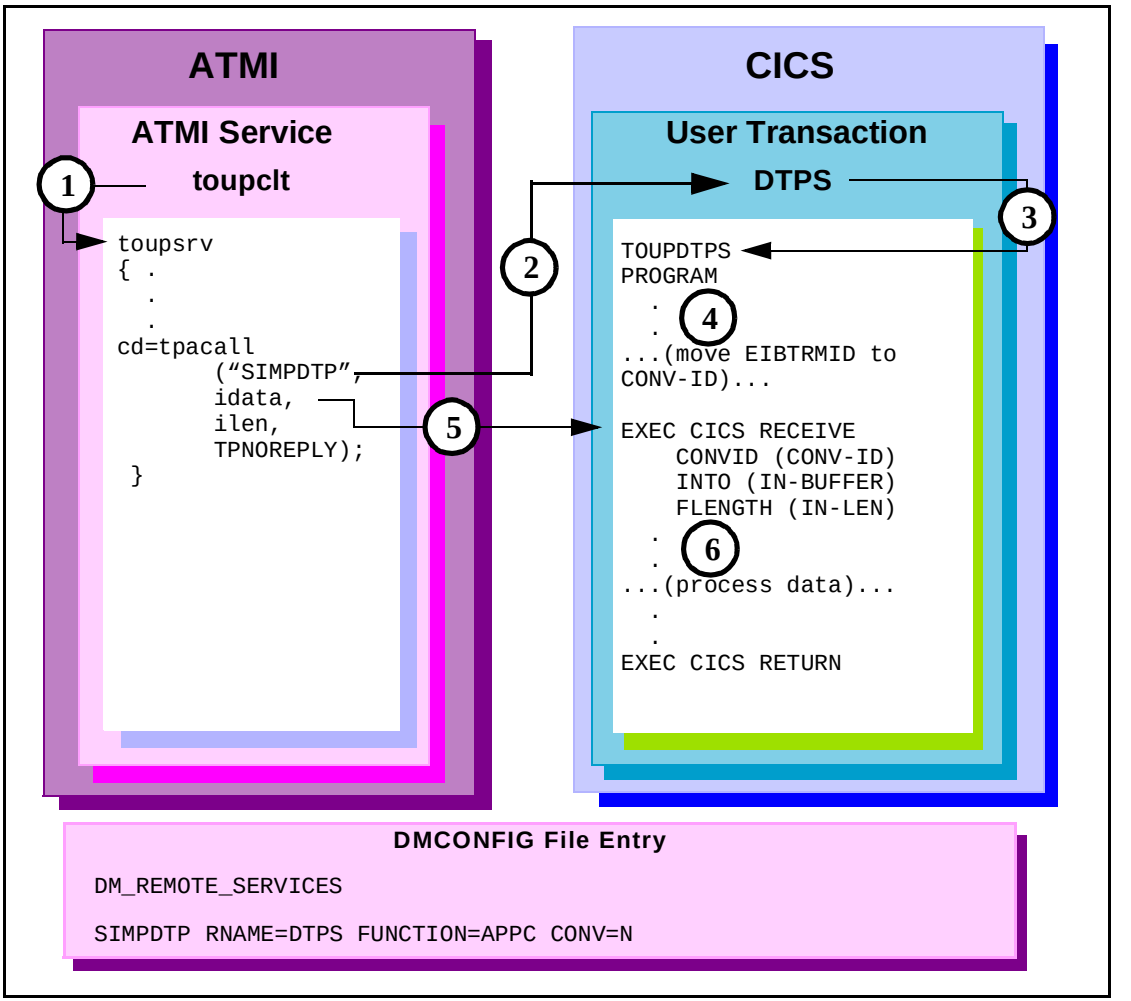
1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpcall` for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of DMCONFIG file.
3. User transaction `DTPS` starts `TOUPDTPS` program.
4. It is recommended you save the `eibtrmid` to a program variable. This value may be used to identify the specific conversation in your CICS/ESA APPC verbs.
5. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.
7. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `odata` buffer. LAST indicates the conversation is finished. WAIT suspends processing until the data has successfully been received.

ATMI Client Asynchronous Request/Response to CICS/ESA DTP



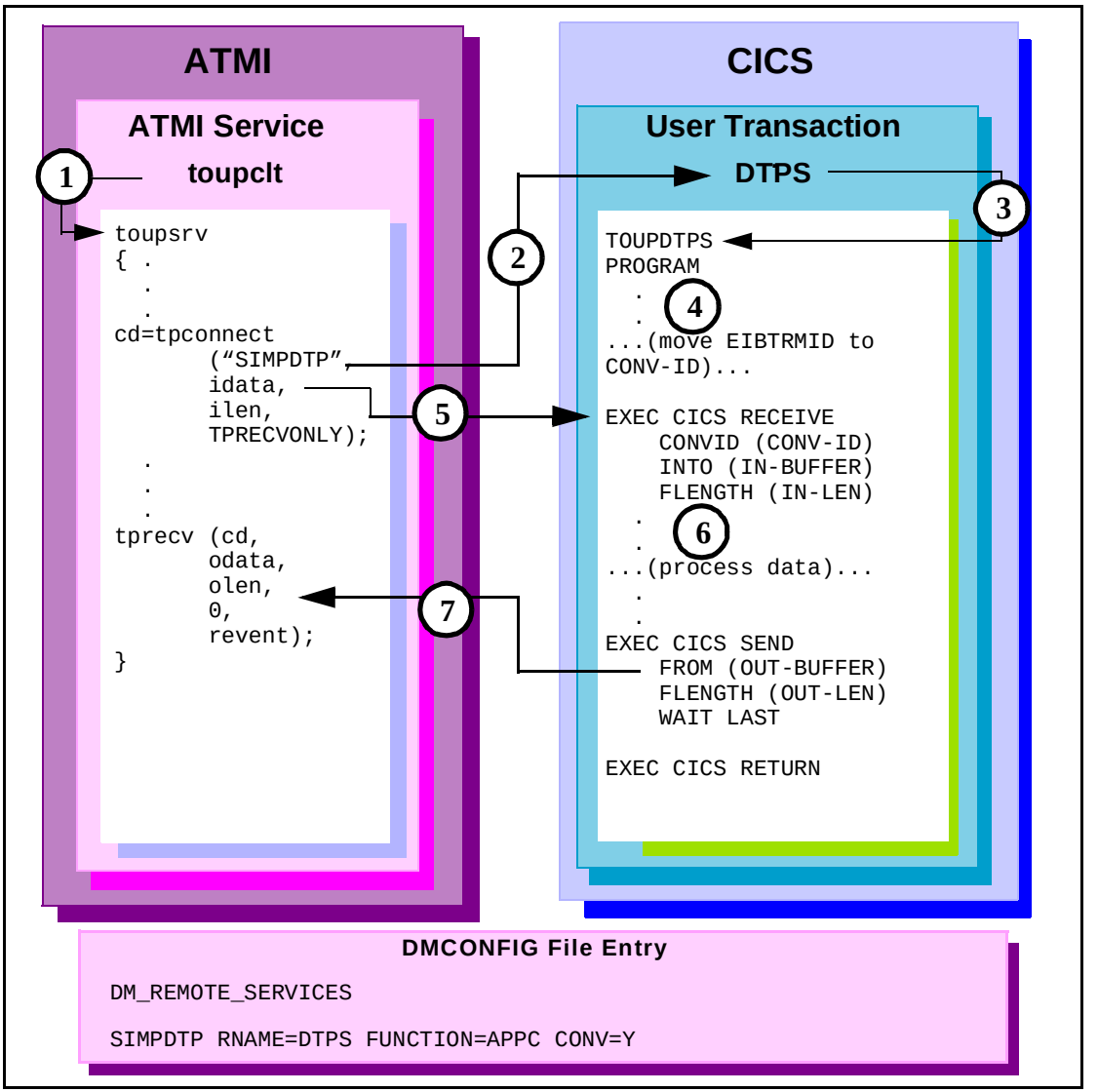
1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpacall` for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of the DMCONFIG file.
3. User transaction DTPS starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation in your CICS/ESA APPC verbs.
5. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.
7. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `tpgetreply odata` buffer. `LAST` indicates the conversation is finished. `WAIT` suspends processing until the data has successfully been received.

ATMI Client Asynchronous Request/Response with No Reply to CICS/ESA DTP



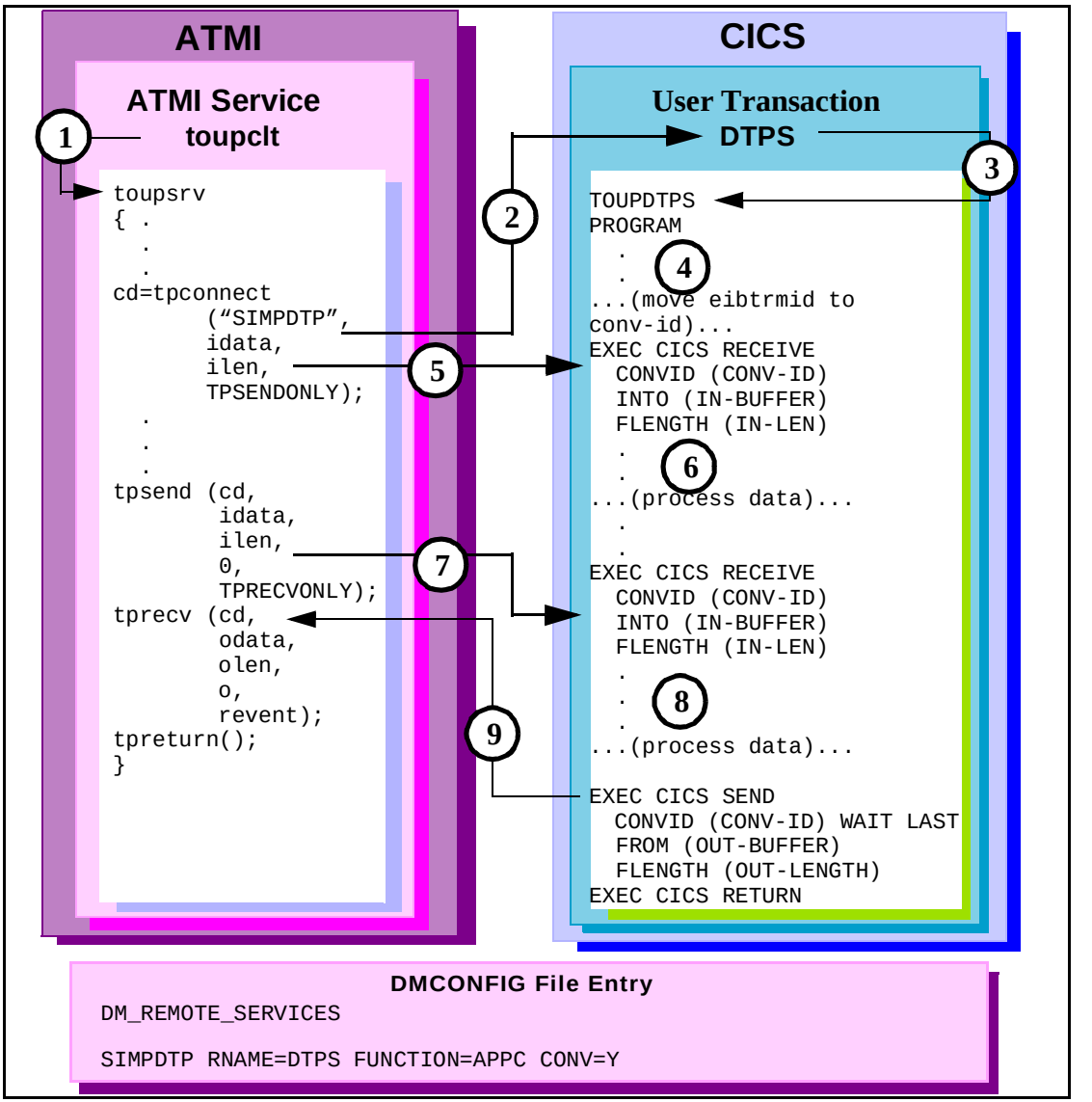
1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpacall` with a `TPNOREPLY` request for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of DMCONFIG file.
3. User transaction DTPS starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your CICS/ESA APPC verbs.
5. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.

ATMI Conversational Client to CICS/ESA DTP, Server Gets Control



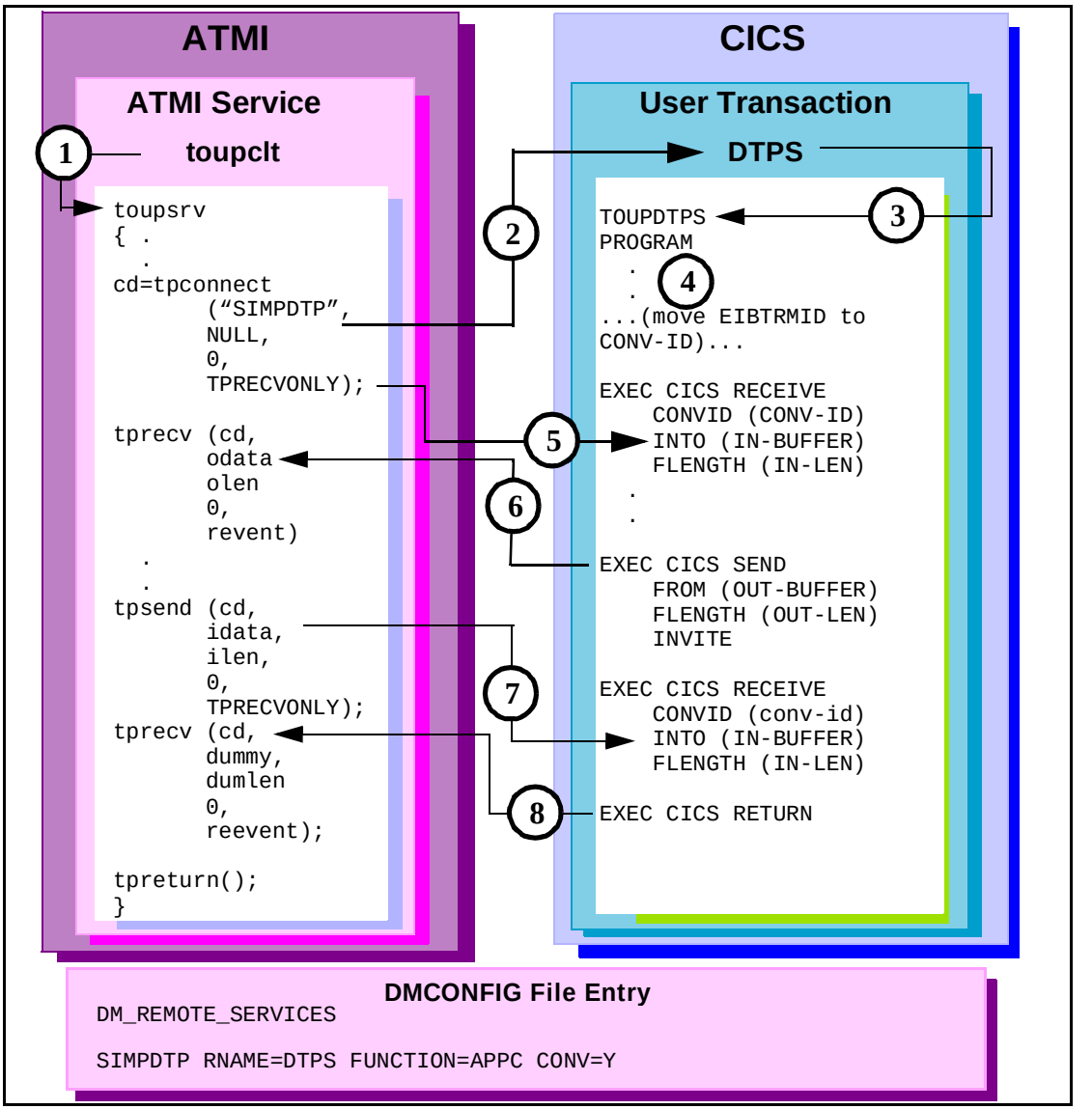
1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpconnect` for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of DMCONFIG file. The `TPRECVONLY` flag indicates the server gets control and the first conversation verb `toupsrv` can issue is `tprecv`. Data is sent on the `tpconnect` in the `idata` buffer.
3. User transaction DTPS starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your CICS/ESA APPC verbs.
5. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.
7. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `tprecv odata` buffer. `WAIT` suspends processing in `TOUPDTPS` until the data has successfully been received. `LAST` indicates the conversation is finished and is communicated to the `tprecv` as `TPEV_SVCSUCC`.

ATMI Conversational Client to CICS/ESA DTP, Client Sends/Receives Data



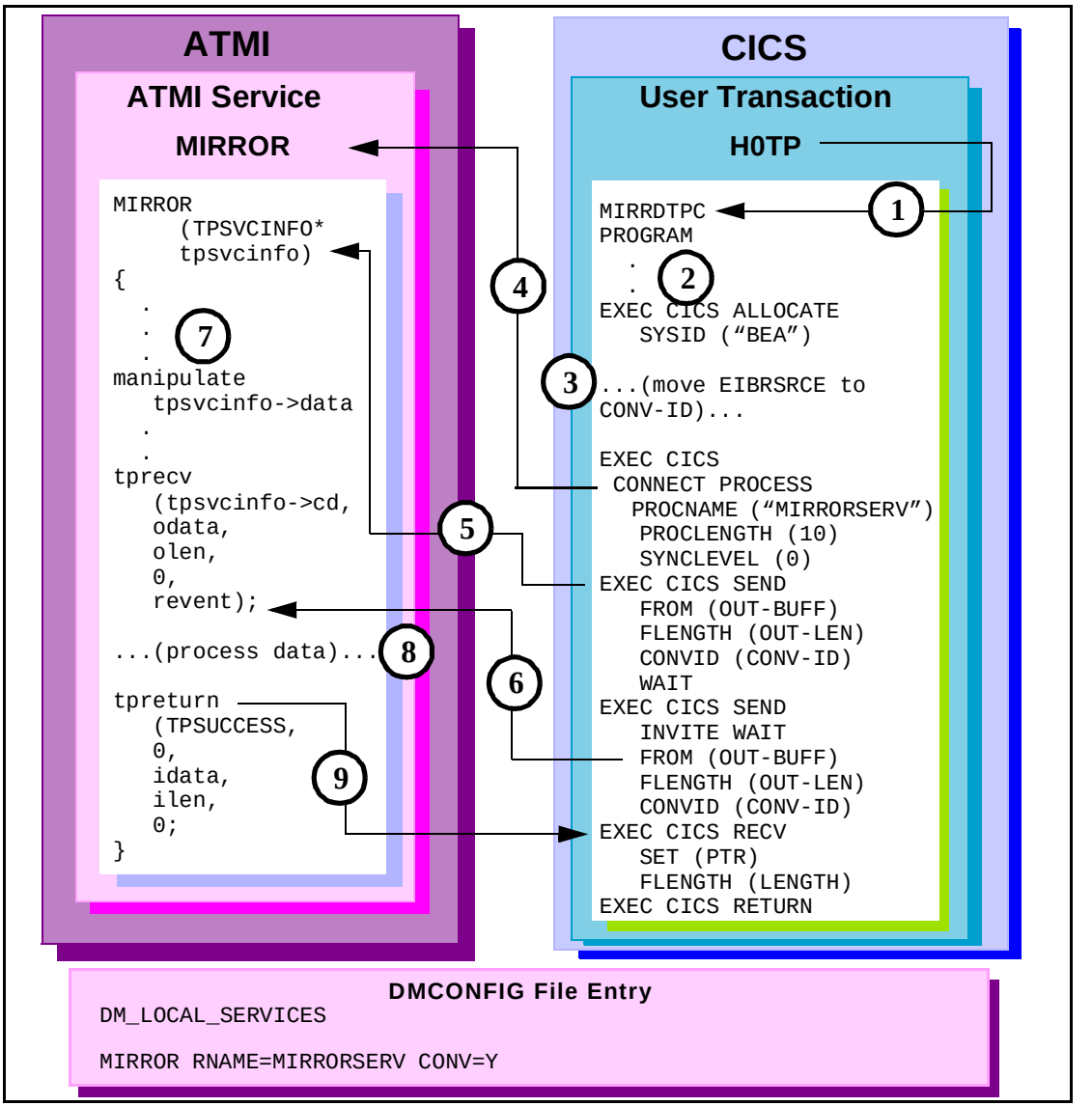
1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpconnect` for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of DMCONFIG file. The `TPSENDONLY` indicates the client retains control and continues to send data. Data is sent on the `tpconnect` in the `idata` buffer.
3. User transaction DTPS starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your CICS/ESA APPC verbs.
5. The `EXEC CICS RECEIVE` command receives the `tpconnect idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.
7. The `EXEC CICS RECEIVE` command receives the `tpsend idata` contents into the server's `IN-BUFFER`.
8. The server processes the data.
9. The `EXEC CICS SEND WAIT LAST` returns `OUT-BUFFER` data in the `tprecv odata` buffer, along with notification that the conversation is over.

ATMI Conversational Client to CICS/ESA DTP, Client Grants Control



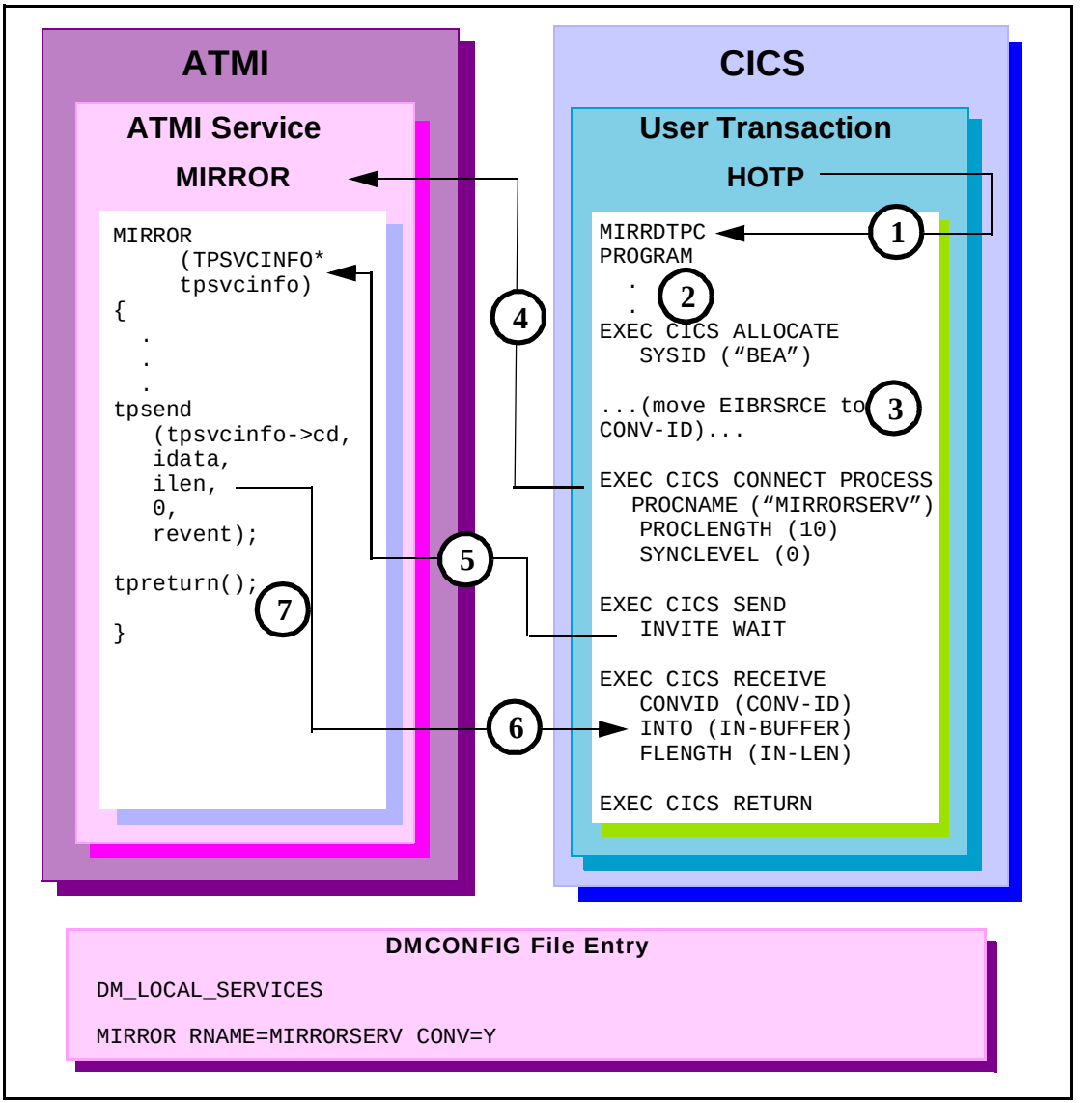
1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpconnect` for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of DMCONFIG file. The `TPRECVONLY` indicates the server gets control and the first conversation verb `toupsrv` can issue is `tprecv`.
3. User transaction DTPS starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your CICS/ESA APPC verbs.
5. The `EXEC CICS RECEIVE` command receives a `send state` indicator from the `tpconnect TPRECVONLY` flag. No data is received into the `INBUFFER`.
6. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `tprecv odata` buffer. The `EXEC CICS SEND` command relinquishes control to the client by using the `INVITE` option. This is communicated to the `tprecv` as `TPEV_SENDONLY`.
7. The `EXEC CICS RECEIVE` command receives the `tpsend idata` contents into the server's `IN-BUFFER`, along with notification that the server has relinquished control.
8. The `EXEC CICS RETURN` ends the conversation, communicated to the `tprecv` as `TPEV_SVCSUCC`.

CICS/ESA DTP to ATMI Conversational Server, Client Retains Control



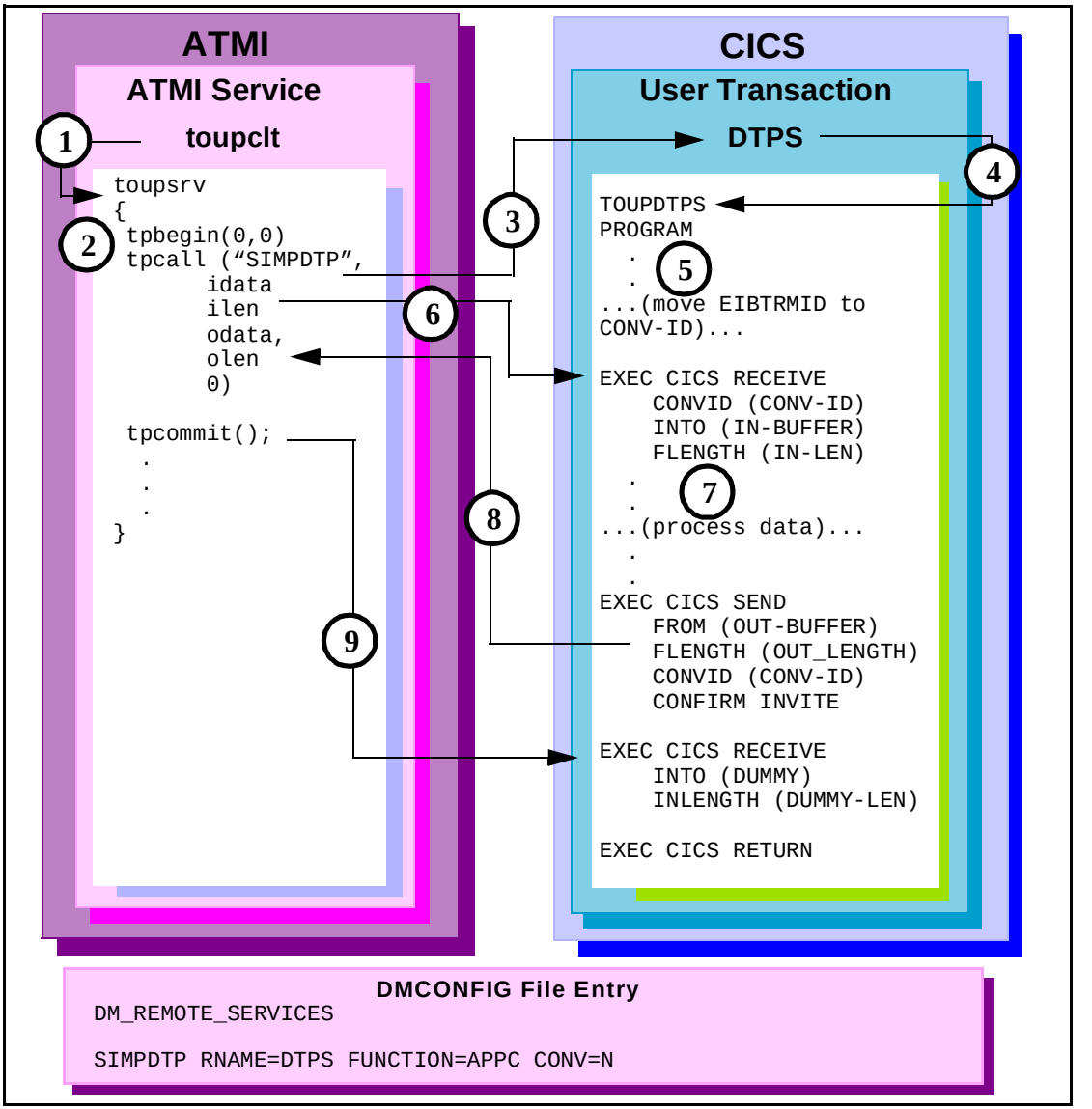
1. User-entered `H0TP` invokes `MIRRDTPC` program.
2. The `EXEC CICS ALLOCATE` acquires a session to the remote Tuxedo domain.
3. Save the conversation ID returned in `EIBRSRCE` to a program variable. This value is used to identify the specific conversation in your CICS/ESA APPC verbs.
4. The `EXEC CICS CONNECT PROCESS` command initiates the advertised service mapped to `MIRROR` in the DM_LOCAL_SERVICES section of the DMCONFIG file.
5. Execute the `EXEC CICS SEND` command to send the contents of the `OUT-BUFFER` to the Tuxedo service in the `tpsvcinfo->data` buffer. The contents might be sent immediately.
6. The `EXEC CICS SEND INVITE WAIT` command sends `out-buff` contents into the `tprecv odata` buffer. The `INVITE` parameter relinquishes control of the conversation, seen as a `TPEV_SENDONLY` in the `reevent` parameter on the `tprecv` command. The data is sent immediately, along with the data from the previous `SEND` operation.
7. The Tuxedo service processes data.
8. The CICS/ESA server processes data.
9. The ATMI `tpreturn` data returns data to the `EXEC CICS RECEIVE`, along with notification that the conversation completed successfully.

CICS/ESA DTP to ATMI Conversational Server, Client Relinquishes Control



1. User-entered HOTP invokes `MIRRDTPC` program.
2. The `EXEC CICS ALLOCATE` acquires a session to the remote Tuxedo domain.
3. Save the conversation ID returned in `EIBRSRCE` to a program variable. This value is used to identify the specific conversation in your CICS/ESA APPC verbs.
4. The `EXEC CICS CONNECT PROCESS` command initiates the advertised service mapped to `MIRROR` in the DM_LOCAL_SERVICES section of the DMCONFIG file.
5. The `EXEC CICS SEND` command relinquishes control with the `INVITE WAIT` option.
6. The `EXEC CICS RECEIVE` command receives the `tpsend idata` buffer contents into the `IN-BUFFER`.
7. The `tpreturn` request tears down the conversation and indicates on the `EXEC CICS RECEIVE` that the conversation is over.

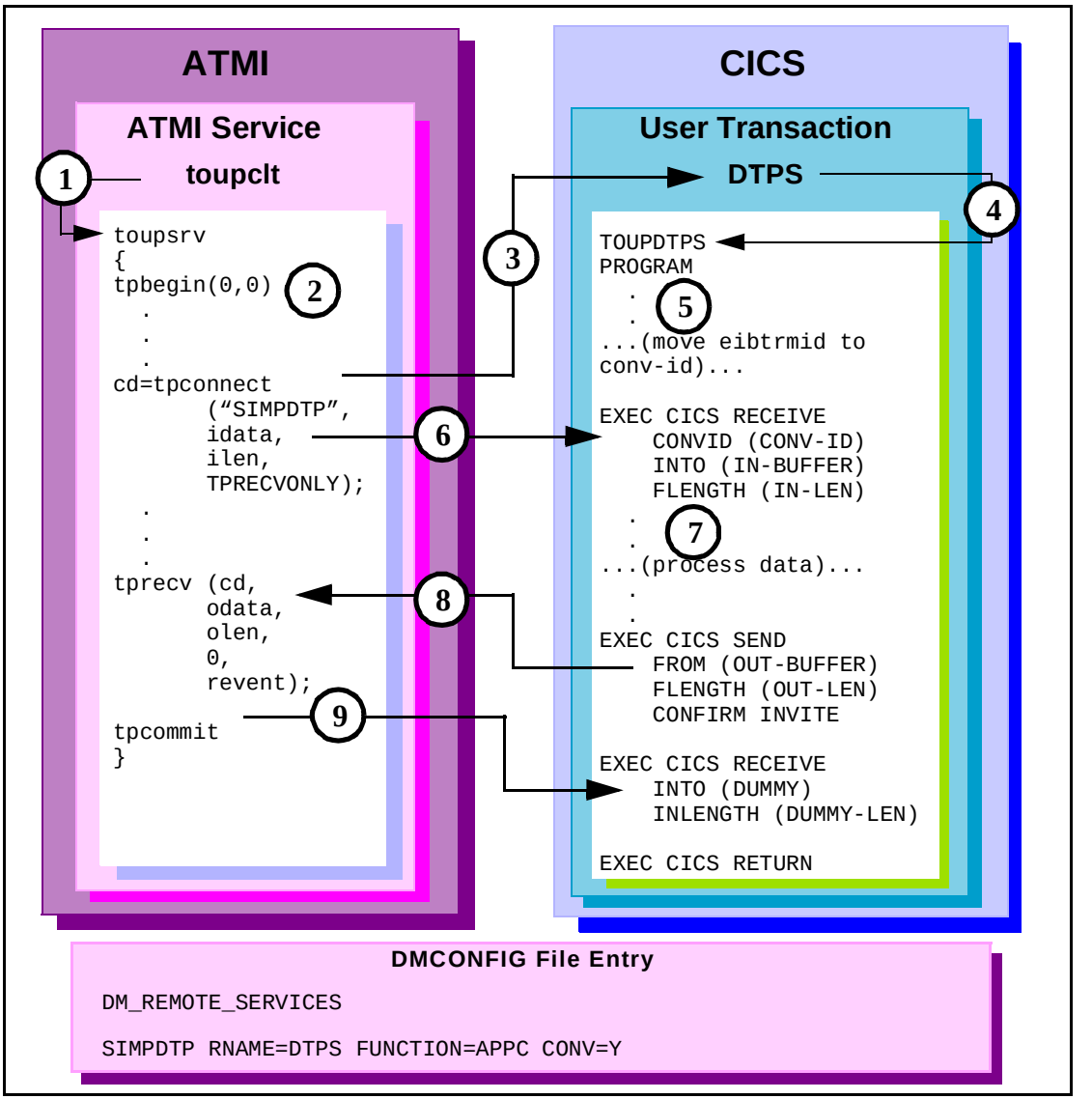
Transactional ATMI Client Request/Response to CICS/ESA DTP



Note: This is not the recommended method of performing a DTP transactional service. Please refer to the transactional DPL using request/response for the recommended method.

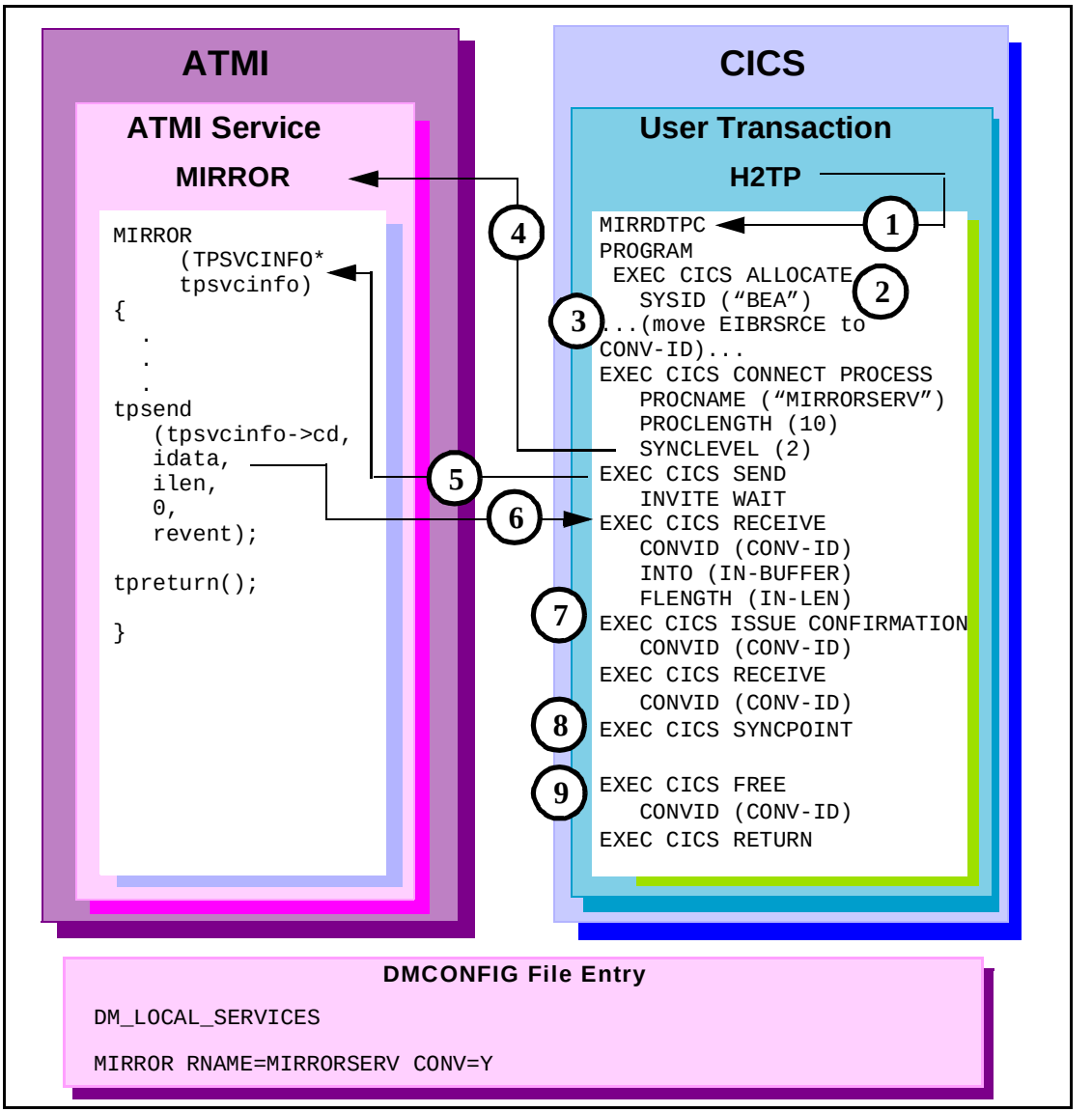
1. ATMI client `toupclt` invokes `toupsrv` service. (Note that each `tpcall` made in the program must be bookended with a `tpbegin` and a `tpcommit`.)
2. The service issues `tpbegin` to start a transaction.
3. The `toupsrv` service issues `tpcall` for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of the DMCONFIG file.
4. User transaction DTPS starts `TOUPDTPS` program.
5. Save the `EIBTRMID` to a program variable. This value is used to identify the specific conversation on your CICS/ESA APPC verbs.
6. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
7. The `TOUPDTPS` program processes data.
8. The `EXEC CICS SEND` command returns the `OUT-BUUFER` contents into the clients `odata` buffer. `CONFIRM` indicates the conversation is finished. `INVITE` allows the client to respond with a `COMMIT` request.
9. The `toupsrv` service issues `tpcommit` to end the transaction. The `COMMIT` is received on the `EXEC CICS RECEIVE` verb and the server issues `EXEC CICS RETURN` to commit the resources, terminate the transaction, and free the outstanding conversation.

Transactional ATMI Conversational Client to CICS/ESA DTP, Server Gets Control



1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.
3. The `toupsrv` service issues `tpconnect` for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of DMCONFIG file. The `TPRECVONLY` indicates the server gains control and the first conversation verb `toupsrv` can issue is `tprecv`. Data is sent on the `tpconnect` in the `idata` buffer.
4. User transaction DTPS starts `TOUPDTPS` program.
5. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your CICS/ESA APPC verbs.
6. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
7. The `TOUPDTPS` program processes data.
8. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `tprecv odata` buffer. `CONFIRM` indicates that the conversation is finished and is communicated to the `tprecv` as `TPEV_SVCSUCC`. `INVITE` enables the client to respond with a `COMMIT` request.
9. The `toupsrv` service issues `tpcommit` to end the transaction. The `COMMIT` is received on the `EXEC CICS RECEIVE` verb and the server issues `EXEC CICS RETURN` to commit the resources, terminate the transaction, and free the outstanding conversation.

Transactional CICS/ESA DTP to ATMI Conversational Server, Host Client Relinquishes Control



1. User-entered H2TP invokes `MIRRDTPC` program.
2. The `EXEC CICS ALLOCATE` acquires a session to the remote Tuxedo domain.
3. Save the conversation ID returned in `EIBRSRCE` to a program variable. This value is used to identify the specific conversation on your CICS/ESA APPC verbs.
4. The `EXEC CICS CONNECT PROCESS` command initiates the advertised service mapped to `MIRRDTPS`. The `SYNCLEVEL(2)` parameter indicates the inclusion of the ATMI service in the CICS/ESA transaction.
5. The `EXEC CICS SEND INVITE WAIT` command causes the client to immediately relinquish control to the Tuxedo server. This is communicated to the service in `TPSVCINFO` as `TPSENDONLY`. No data is sent to the server on this request.
6. The `EXEC CICS RECEIVE` command receives the `tpsend` idata buffer contents into the `IN-BUFFER`. The `EXEC CICS RECEIVE` command receives a confirm request indicating the conversation should be terminated.
7. The `EXEC CICS ISSUE CONFIRMATION` verb responds positively to the confirm request.
8. The `EXEC CICS SYNCPOINT` is an explicit commit request to end the conversation and update all resources in the transaction.
9. The `EXEC CICS FREE` verb explicitly frees the outstanding conversation.

CPI-C Programming Examples

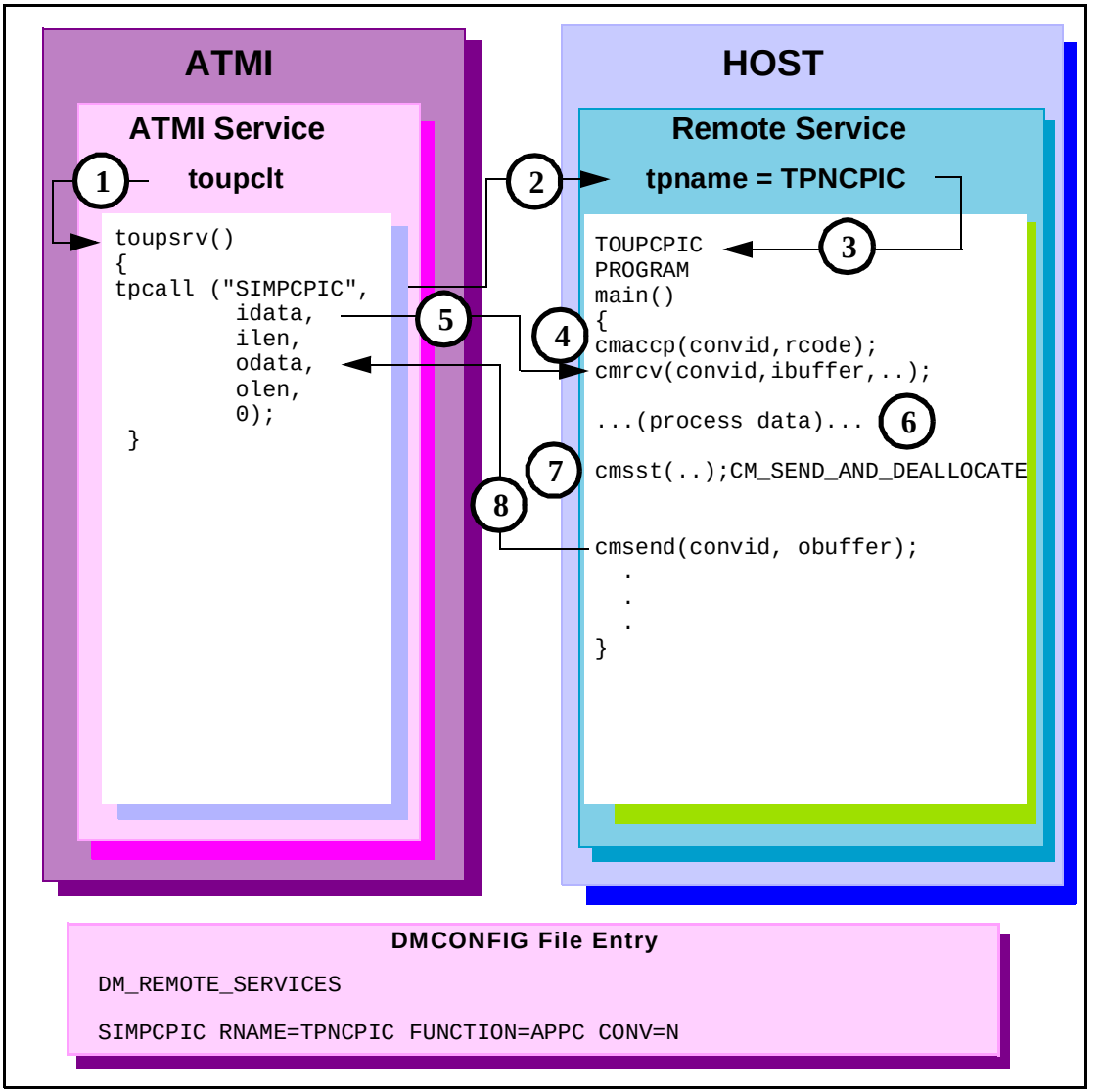
The examples in this section show the protocol exchanges between the local ATMI platform and remote host application program. The type of ATMI service request determines the nature of the client/server communication model. For requests initiated by the host application, the configuration information for the local service determines the protocol exchanges on the conversation.

Although it is most suited for the DPL environment, the `tpcall` is usually used in the DPL environment but can also be used for a request/response to an APPC server.

The examples in this section represent a few of the many programming scenarios available for using CPI-C and ATMI service invocations. These examples employ the most natural and efficient approaches.

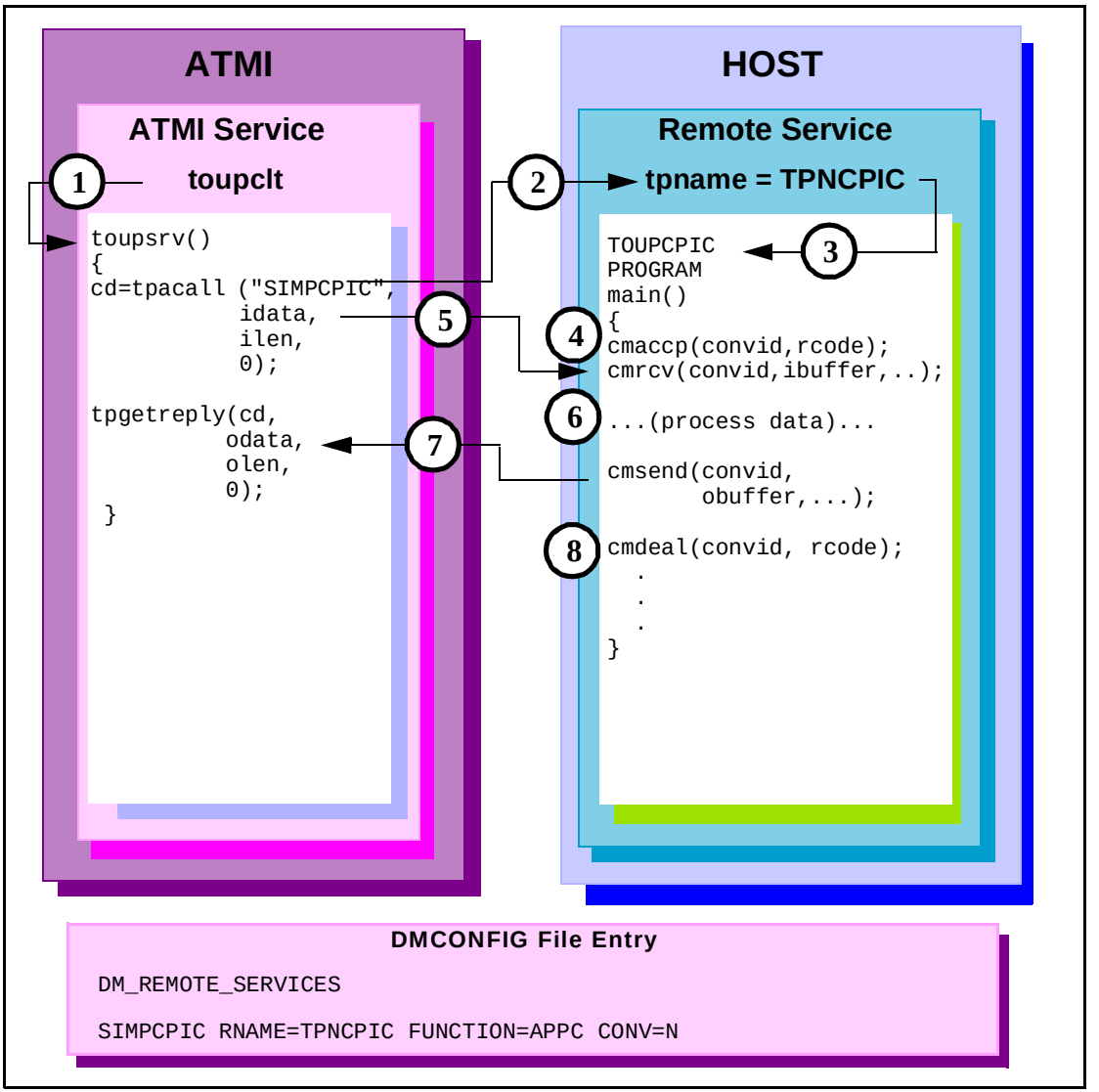
Note: To run transactional client/server scenarios or the CPI Resource Recovery interface, the eAM software must be licensed for sync-level 2 operations.

ATMI Client Request/Response to Host CPI-C



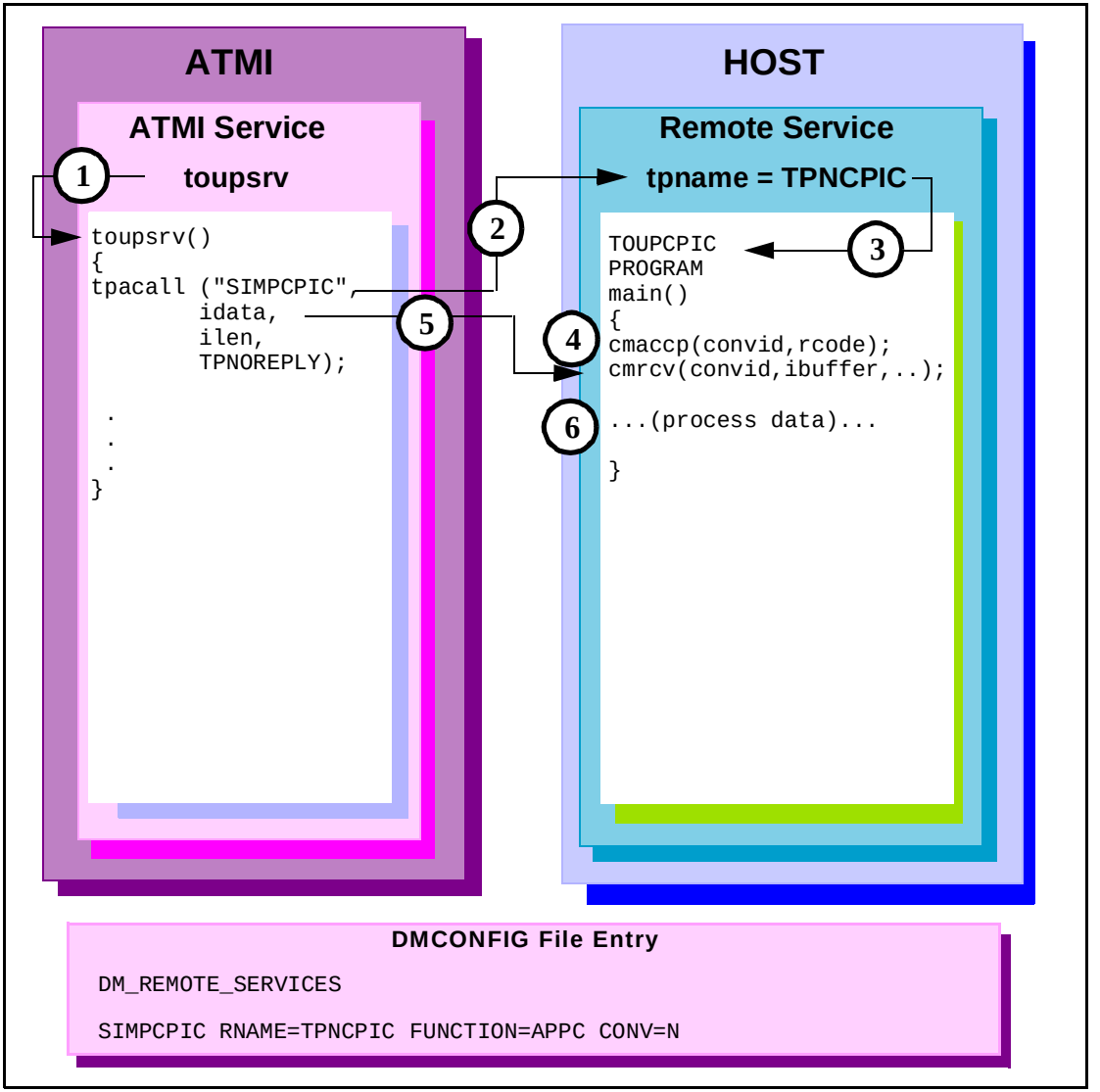
1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpcall` for `SIMPCPIC`, which is advertised in the DM_REMOTE_SERVICES section of the DMCONFIG file.
3. The remote service with the `tpname` `TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `idata` buffer contents for processing
6. The `TOUPCPIC` program processes data.
7. The `cmsst` request prepares the next send request by setting the send type to `CM_SEND_AND_DEALLOCATE`.
8. The `cmsend` request returns the `obuffer` contents into the client `odata` buffer. The buffer is flushed, and the conversation ended.

ATMI Client Asynchronous Request/Response to Host CPI-C



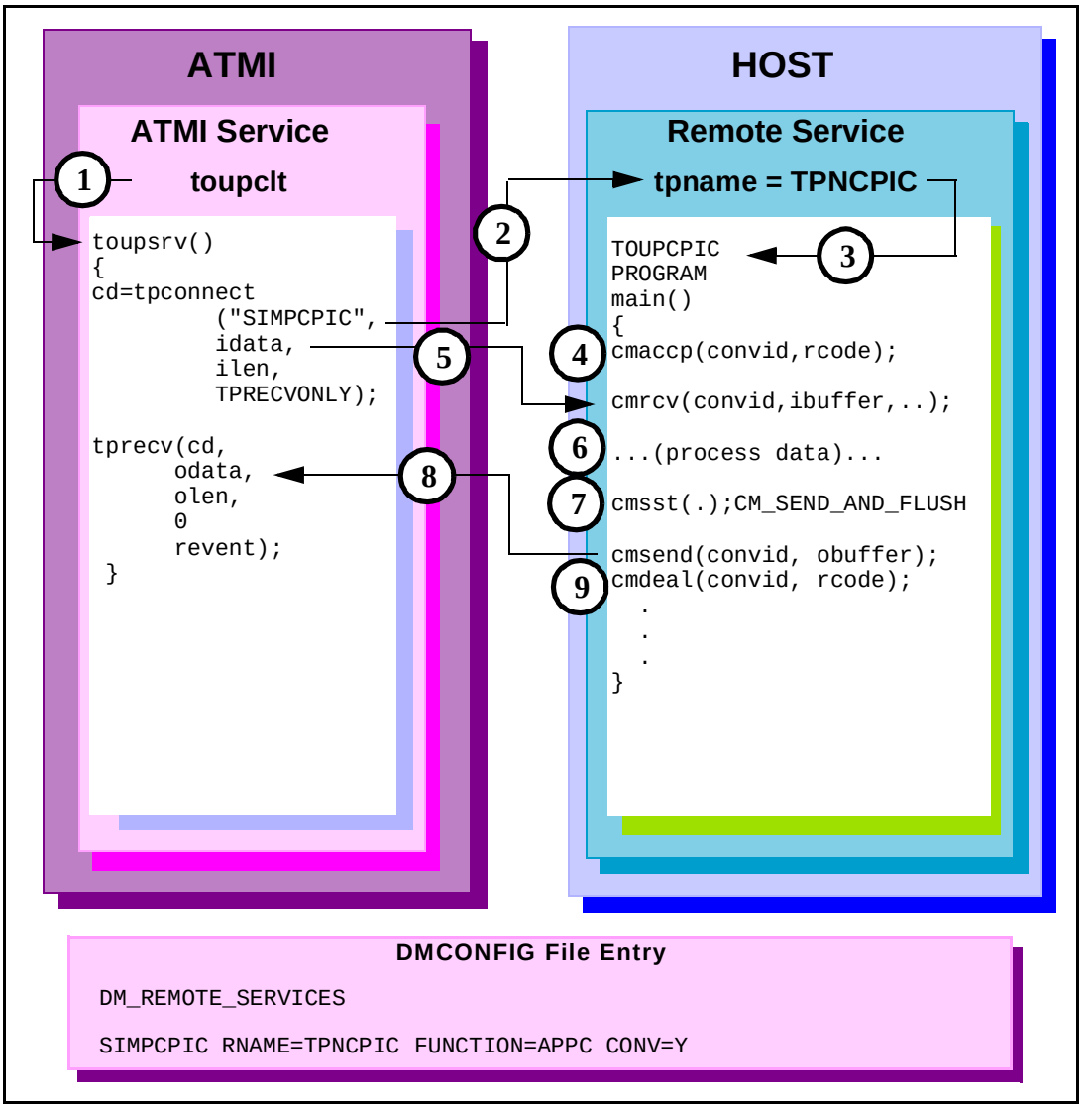
1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpacall` for `SIMPCPIC`, which is advertised in the DM_REMOTE_SERVICES section of the DMCONFIG file.
3. The remote service with `tpname TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `idata` buffer contents for processing.
6. The `TOUPCPIC` program processes data.
7. The `cmsend` command returns the `obuffer` contents into the client `tpgetreply odata` buffer. The data may not be immediately sent to the `tpgetreply odata` buffer on this request.
8. The `cmdeal` flushes the data to the client, and indicates the conversation is finished.

ATMI Client Asynchronous Request/Response to Host CPI-C with No Reply



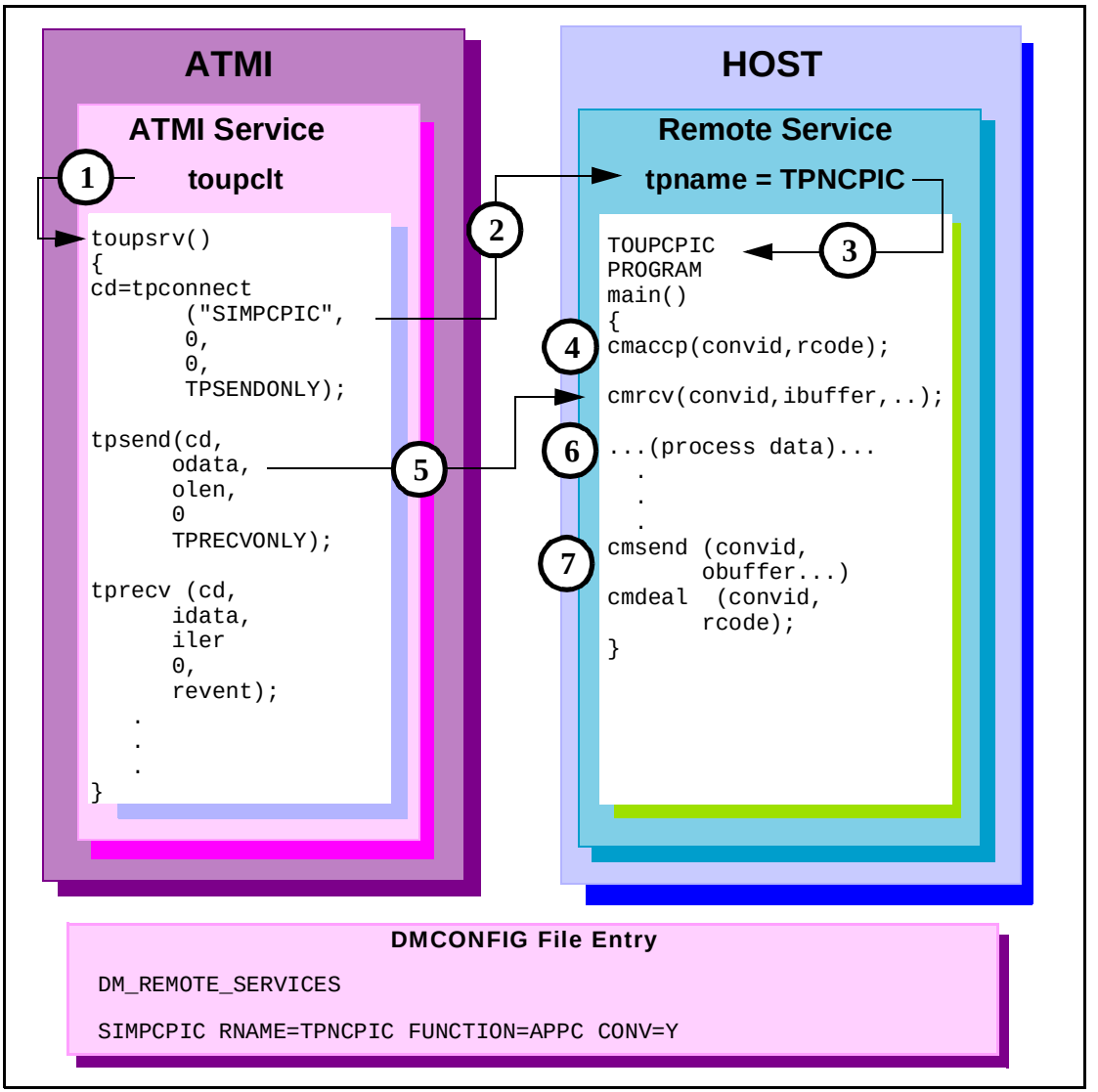
1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues tpacall with a `TPNOREPLY` request for `SIMPCPIC`, which is advertised in the DM_REMOTE_SERVICES section of the DMCONFIG file.
3. The remote service with `tpname TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `idata` buffer contents for processing, and notification that the conversation has ended with the return code value of `CM_DEALLOCATED_NORMAL`.
6. The `TOUPCPIC` program processes data.

ATMI Conversational Client to Host CPI-C, Server Gets Control



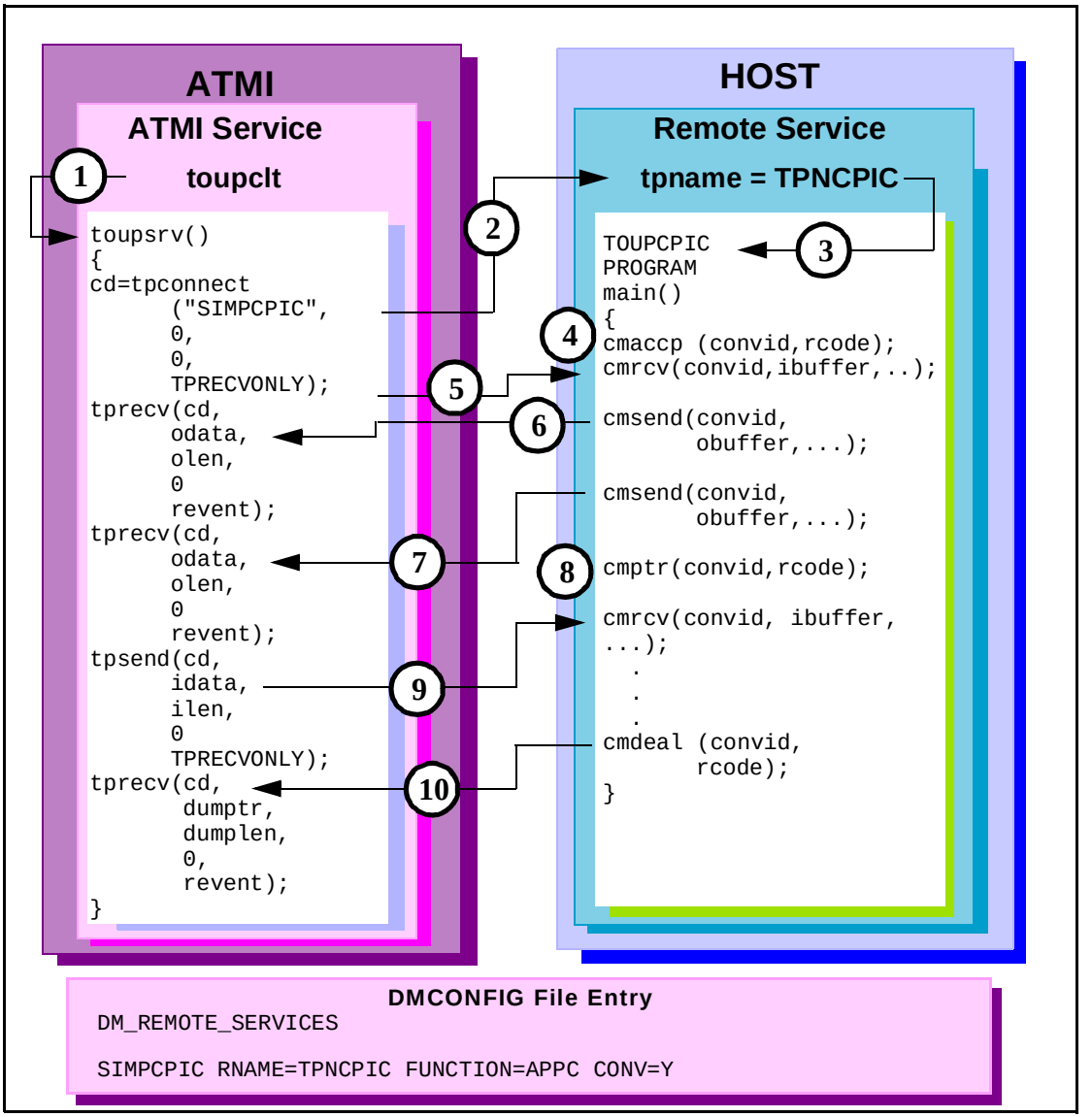
1. ATMI client invokes `toupsrv` service
2. The `toupsrv` service issues `tpconnect` for `SIMPCPIC`, which is advertised in the DM_REMOTE_SERVICES section of the DMCONFIG file. The `TPRECVONLY` indicates the server gains control and the first conversation verb the `toupsrv` can issue is `tprecv`. Data is sent on the `tpconnect` in the `idata` buffer.
3. The remote service with `tpname TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation ID returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `idata` buffer contents for processing.
6. The `TOUPCPIC` program processes data
7. The `cmsst` request prepares the next send request by setting the send type to `CM_SEND_AND_FLUSH.`
8. The `cmsend` command returns the `obuffer` contents into the client `tprecv odata` buffer. The data is immediately flushed on the send request.
9. The `cmdeal` request ends the conversation.

ATMI Conversational Client To Host CPI-C, Client Retains Control



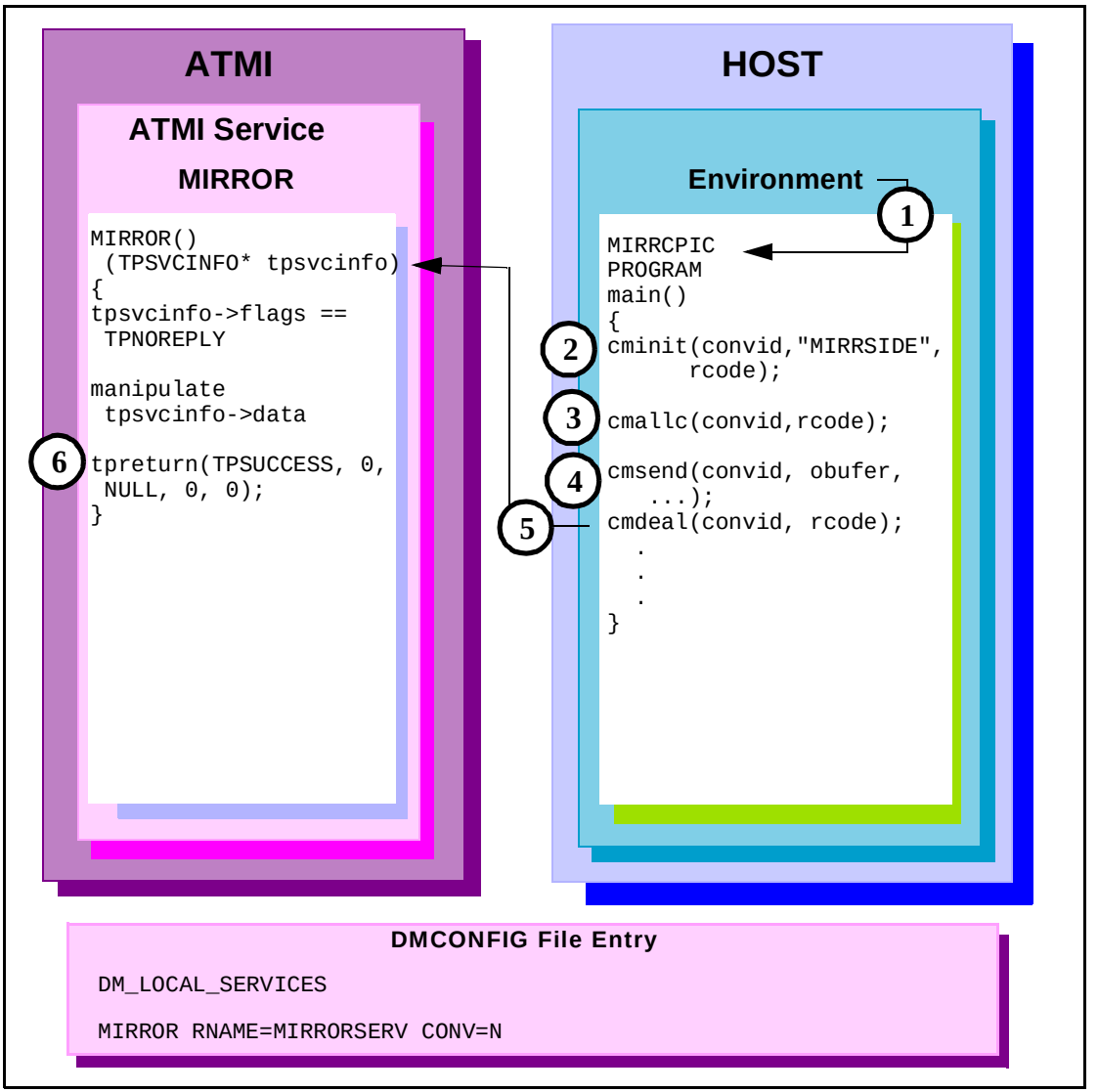
1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpconnect` for `SIMPCPIC`, which is advertised in the DM_REMOTE_SERVICES section of the DMCONFIG file. The TPSENDONLY indicates the client retains control and continues to send data. No data is sent with the `tpconnect`.
3. The remote service with `tpname TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `tpsend idata` buffer contents for processing. The conversation is relinquished with the `TPRECVONLY` flag.
6. The `TOUPCPIC` program processes data.
7. The `cmsend` returns a response in the `tprecv idata` buffer, along with notification from the `cmdeal` command that the conversation is over. The `cmdeal` flushes the data buffer and the `tprecv reevent` parameter is set to `TPEV_SUCCESS`.

ATMI Conversational Client to Host CPI-C, Client Grants/gets Control



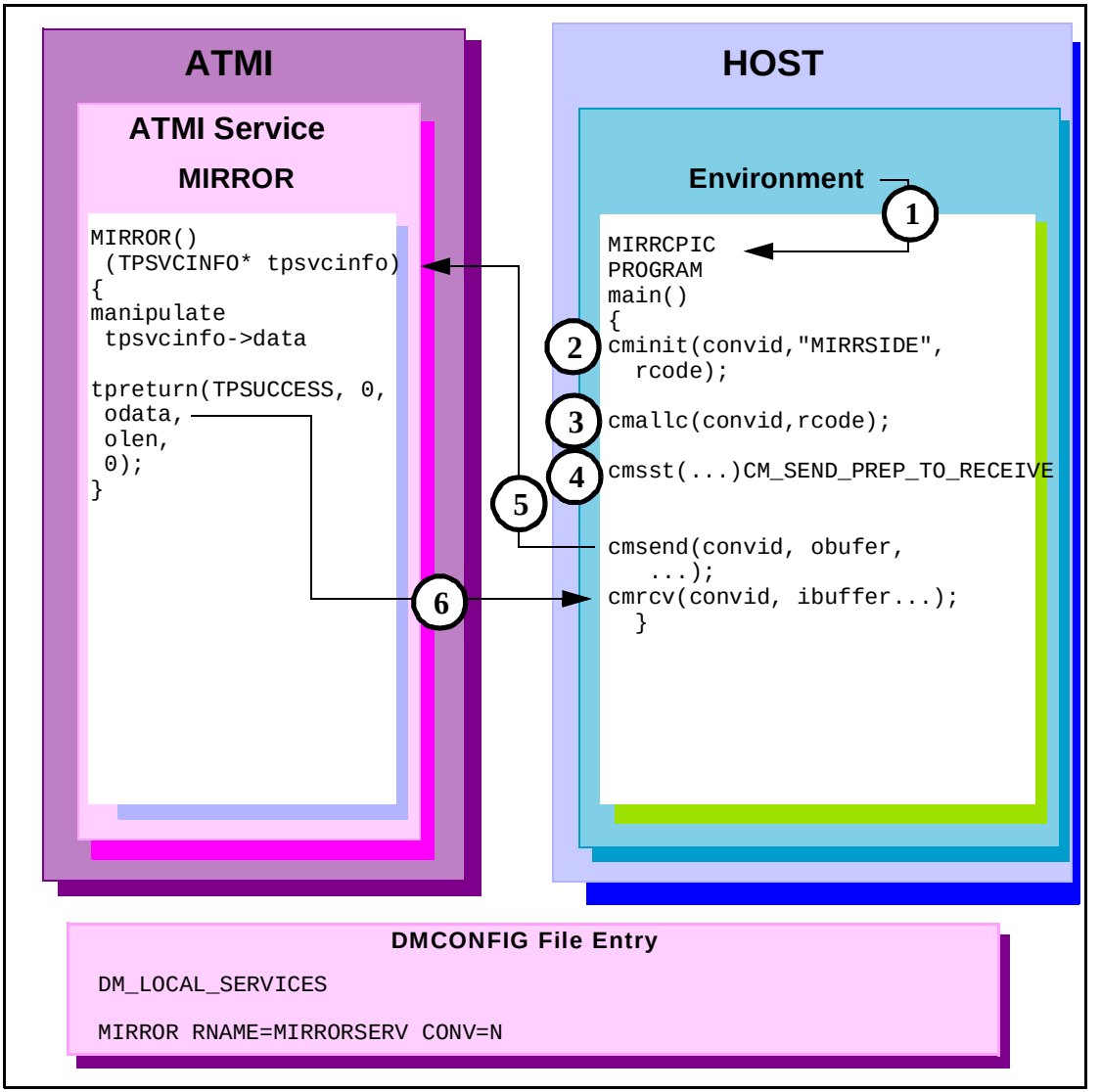
1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpconnect` for `SIMPCPIC`, which is advertised in the DM_REMOTE_SERVICES section of the DMCONFIG file. The TPRECVONLY indicates the server gains control and the first conversation verb the `toupsrv` can issue is `tprecv`.
3. The remote service with `tpname TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` request. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` requests receives the indicator that control has been granted to the server.
6. The `cmsend` request returns its `obuffer` contents into the first client `tprecv odata` buffer. The data may not immediately be sent.
7. The `cmsend` request returns its `obuffer` contents into the second client `tprecv odata` buffer. The data may not immediately be sent.
8. The `cmptr` request flushes the data to the client, and grants control to the client.
9. The `cmrcv` request receives the `tpsend idata` buffer contents for processing. The `TPRECVONLY` is passed to the `tprecv`, relinquishing control of the conversation.
10. The `cmdeal` indicates a successful completion of the conversation to the `tprecv`; no data is passed.

Host CPI-C to ATMI Asynchronous Request/Response Server with No Reply



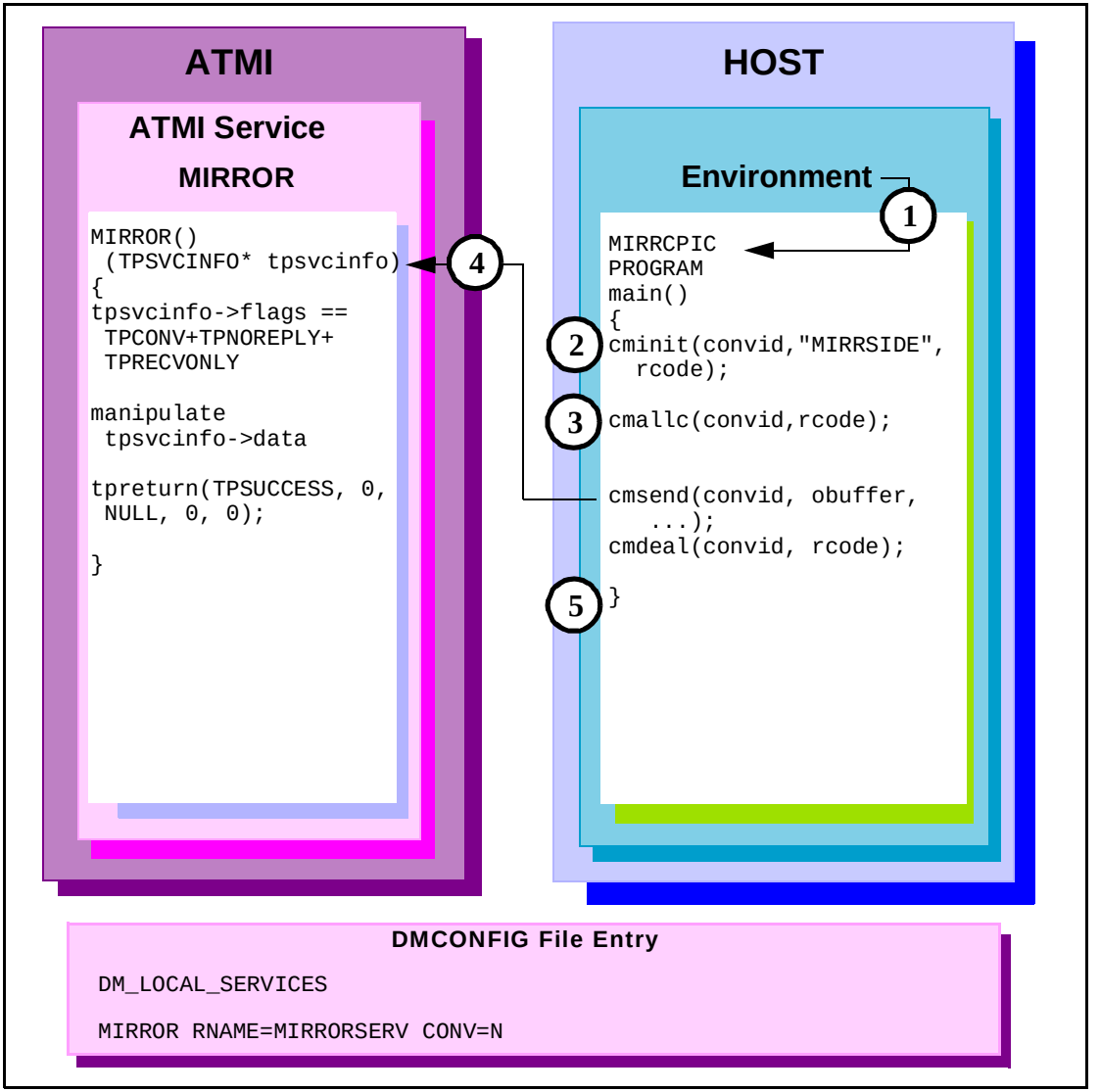
1. The CPI-C application program `MIRRCPIC` is invoked using environment start-up specifications.
2. The `MIRRCPIC` client requests `cminit` to establish conversation attributes and receive a conversation ID that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cmallc` request initiates the advertised service mapped to `MIRRORSERV` in the DM_LOCAL_SERVICES section of the DMCONFIG file.
4. The `cmsend` request sends the contents of `obuffer` to the ATMI service in the `tpsvcinfo->data` buffer.
5. The `cmdeal` request flushes the data, and indicates the conversation is finished with the TPNOREPLY in the `tpsvcinfo->flag` field.
6. The service completes with the `tpreturn`.

Host CPI-C to ATMI Server Request/Response



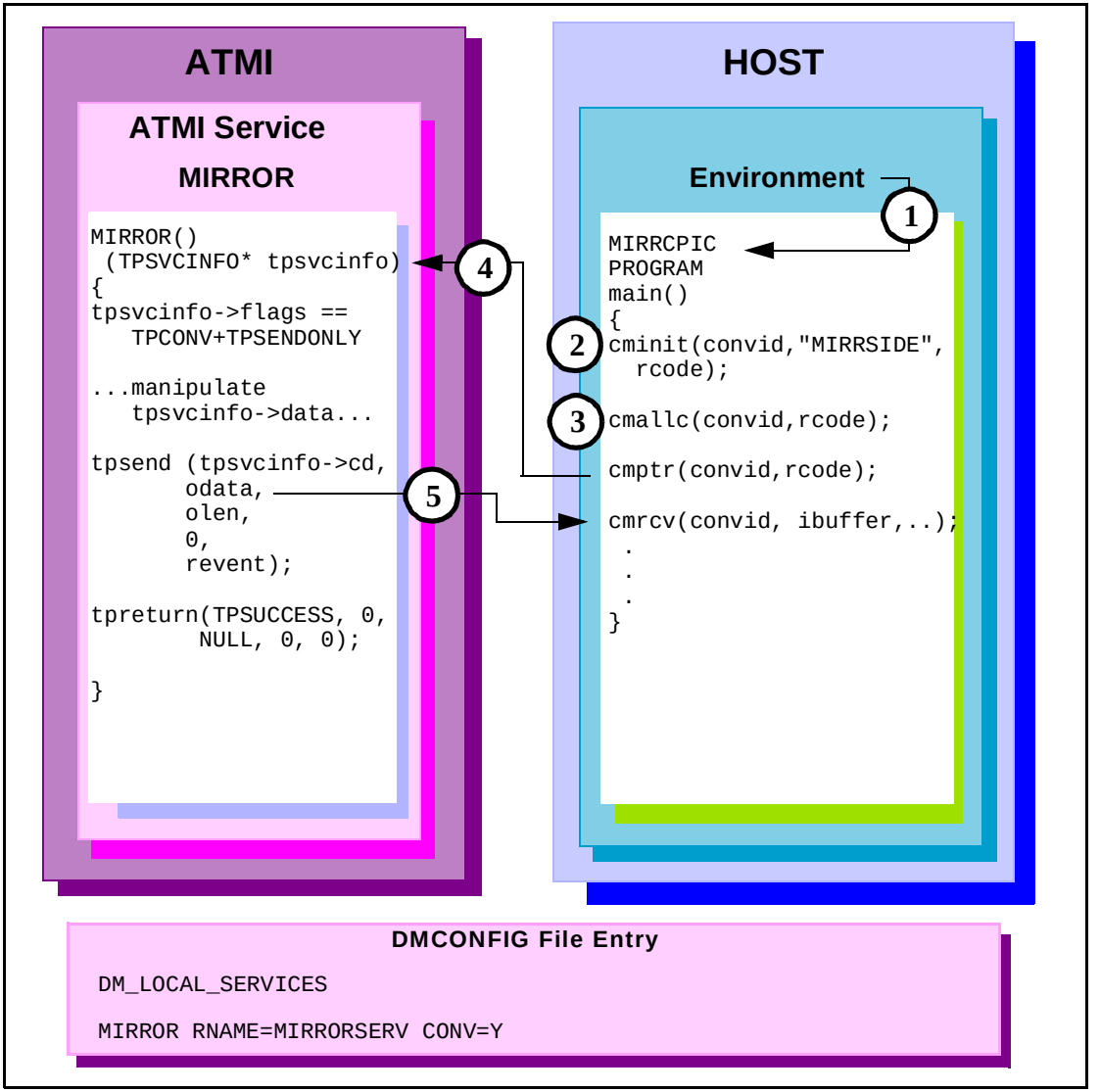
1. The CPI-C application program MIRRCPIC is invoked using environment start-up specifications.
2. The `MIRRCPIC` client requests `cminit` to establish conversation attributes and receive a conversation id that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cmallc` request initiates the advertised service mapped to `MIRRORSERV` in the DM_LOCAL_SERVICES section of the DMCONFIG file.
4. The `cmsst` request prepares the next send request by setting the send type to `CM_SEND_AND_PREP_TO_RECEIVE`.
5. The `cmsend` request immediately sends the contents of `obuffer` to the ATMI service in the `tpsvcinfo->data` buffer and relinquishes control to the `mirrorserv` service.
6. The `cmrcv` request receives the contents of the `odata` returned on the ATMI `tpreturn` service, and notification that the conversation has ended with the return code value of `CM_DEALLOCATED_NORMAL`.

Host CPI-C to ATMI Conversational Service, Client Retains Control



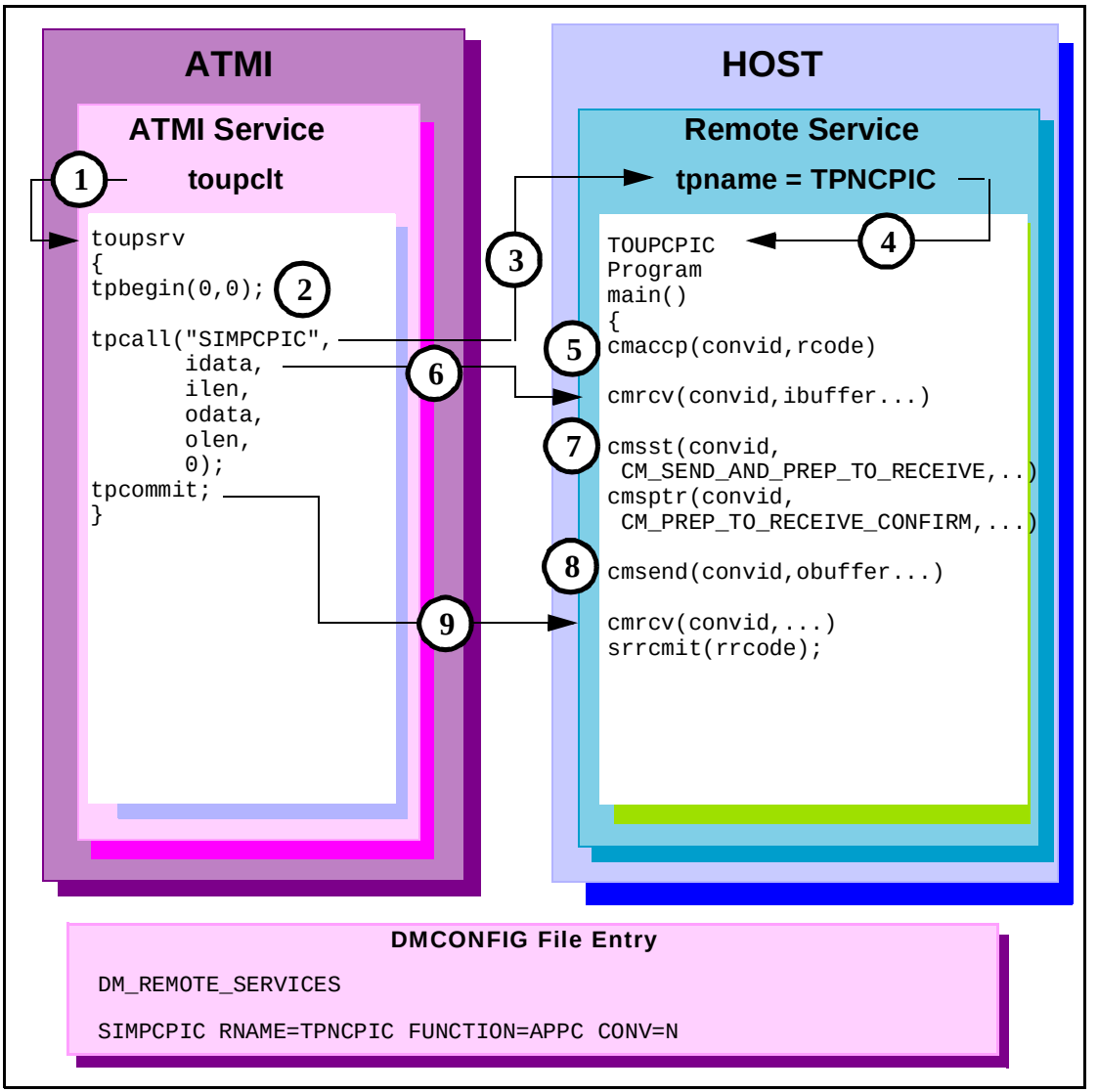
1. The CPI-C application program MIRRCPIC is invoked using environment start-up specifications.
2. The `MIRRCPIC` client requests `cminit` to establish conversation attributes and receive a conversation id that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cmallc` request initiates the advertised service mapped to `MIRRORSERV` in the DM_LOCAL_SERVICES section of the DMCONFIG file.
4. The `cmsend` request sends the contents of `obuffer` to the ATMI service in the `tpsvcinfo->data` buffer.
5. The `cmdeal` request flushes the data and ends the conversation, as indicated by `TPNOREPLY` in the `tpsvcinfo->flag` field.

Host CPI-C ATMI to Conversational Service, Client Grants Control



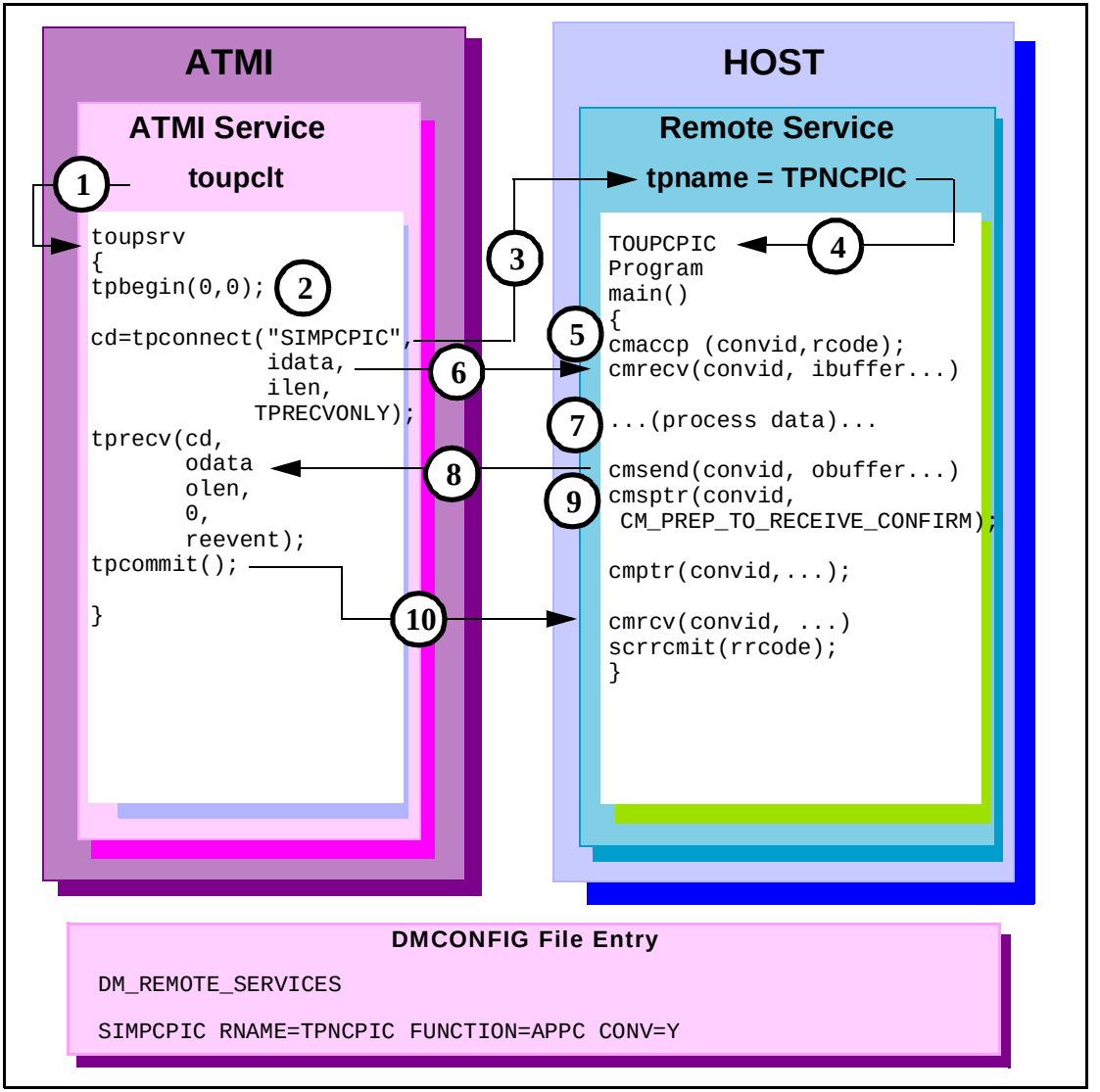
1. The CPI-C application program `MIRRCPIC` is invoked using environment start-up specifications.
2. The `MIRRCPIC` client requests `cminit` to establish conversation attributes and receive a conversation ID that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cmallc` request initiates the advertised service mapped to `MIRROR` in the DM_LOCAL_SERVICES section of the DMCONFIG file.
4. The `cmptr` relinquishes control of the conversation to the ATMI service indicated as `TPSENDONLY` in the `tpsvcinfo->flag` field. No data is passed in the `tpsvcinfo->data` field.
5. The `cmrcv` receives the contents of the `tpsend odata` into the `ibuffer`. The end of the conversation is passed from the `tpreturn` service as return code value `CM_DEALLOCATED_NORMAL`.

Transactional ATMI Client Request/Response to Host CPI-C



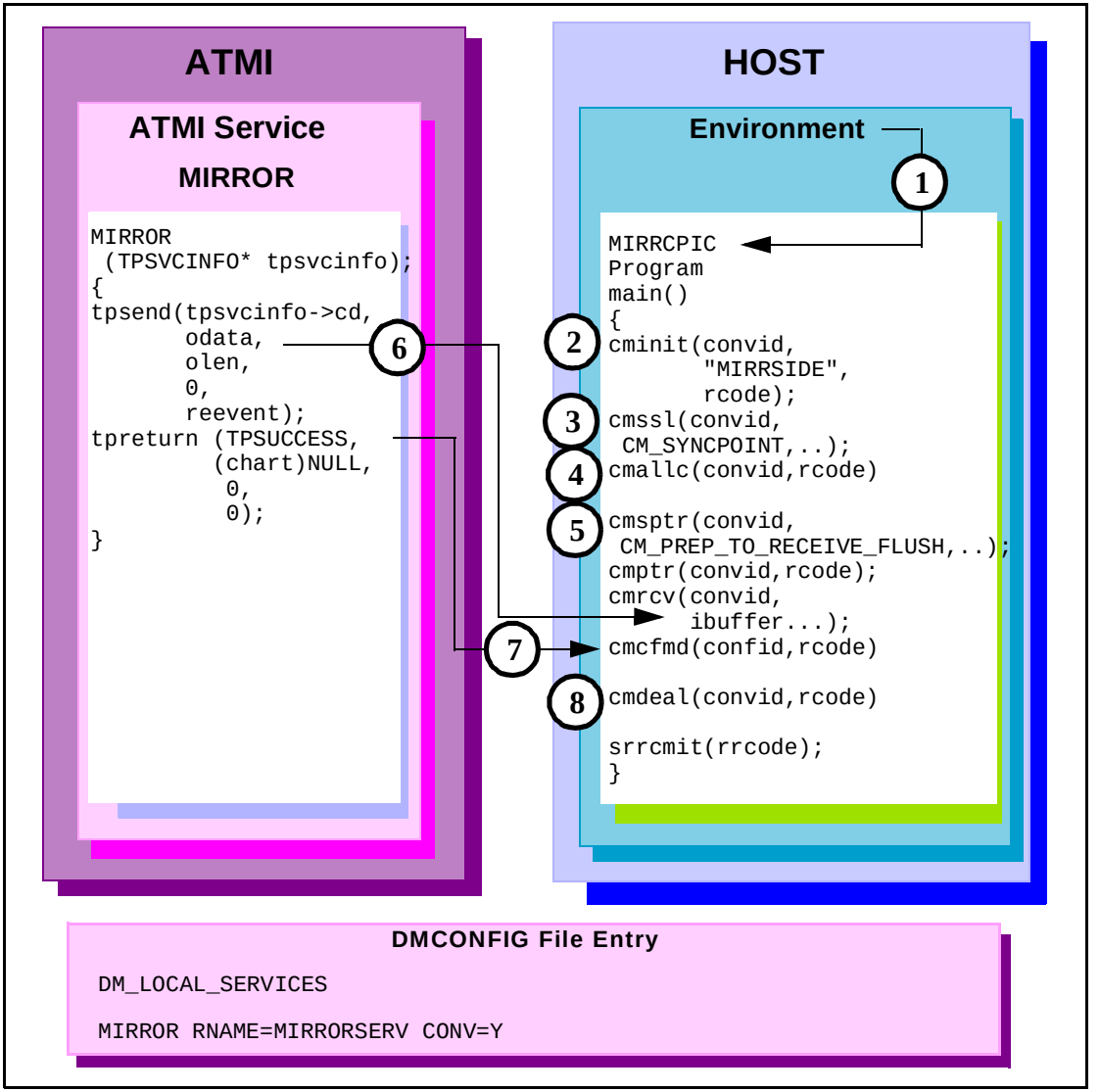
1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.
3. The `toupsrv` service issues `tpcall` for `SIMPCPIC`, which is advertised in the DM_REMOTE_SERVICES section of the DMCONFIG file. Data is sent from the `idata` buffer on the `tpconnect`.
4. The remote service with `tpname TPNCPIC` invokes `TOUPCPIC` program.
5. The server accepts the conversation with the `cmaccp` call. The conversation ID returned on the request in `convid` is used for all other requests during this conversation.
6. The `cmrcv` request receives the `idata` buffer contents for processing.
7. The `cmsst` and `cmsptr` prepare the next send request by setting the send type to `CM_SEND_AND_PREP_TO_RECEIVE` and by setting the prepare-to-receive type to `CM_PREP_TO_RECEIVE_CONFIRM`.
8. The `cmsend` request immediately returns the `obuffer` contents into the client's `odata` buffer. The server relinquishes control to the server and indicates the end of the conversation with the `CONFIRM` request.
9. The `toupsrv` issues the `tpcommit` to successfully complete the transaction and commit all updated resources. The `cmrcv` request receives the commit request, and responds explicitly to the request with the SAA Resource/Recovery commit call `srrcmit`. The conversation is ended after the successful commit exchange.

Transactional ATMI Conversational Client to Host CPI-C, Server Gets Control



1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.
3. The `toupsrv` service issues a `tpconnect` service request for `SIMPCPIC`, which is advertised in the DM_REMOTE_SERVICES section of the DMCONFIG file. Data is sent in the `idata` buffer on the `tpconnect`.
4. The remote service with `tpname TPNCPIC` invokes `TOUPCPIC` program.
5. The server accepts the conversation with the `cmaccp` call. The conversation ID returned on the request in `convid` is used for all other requests during this conversation.
6. The `cmrcv` request receives the `idata` buffer contents sent on the `tpconnect` for processing.
7. The `TOUPCPIC` program processes the data.
8. The `cmsend` returns the `obuffer` contents into the client's `tprecv odata` buffer. The buffer contents may not be sent immediately.
9. The `cmsptr` prepares the prepare-to-receive request with `CM_PREP_TO_RECEIVE_CONFIRM`. The `cmptr` request with `CONFIRM` indicates that the conversation is finished and is communicated to the `tprecv` as `TPEV_SVCSUCC`.
10. The `toupsrv` issues the `tpcommit` to successfully complete the transaction and commit all updated resources. The `cmrcv` request receives the commit request and responds explicitly to the request with the SAA Resource/Recovery commit call `srrcmit`. The conversation is ended after the successful commit exchange.

Transactional Host CPI-C to ATMI Conversational Server, Client Grants Control

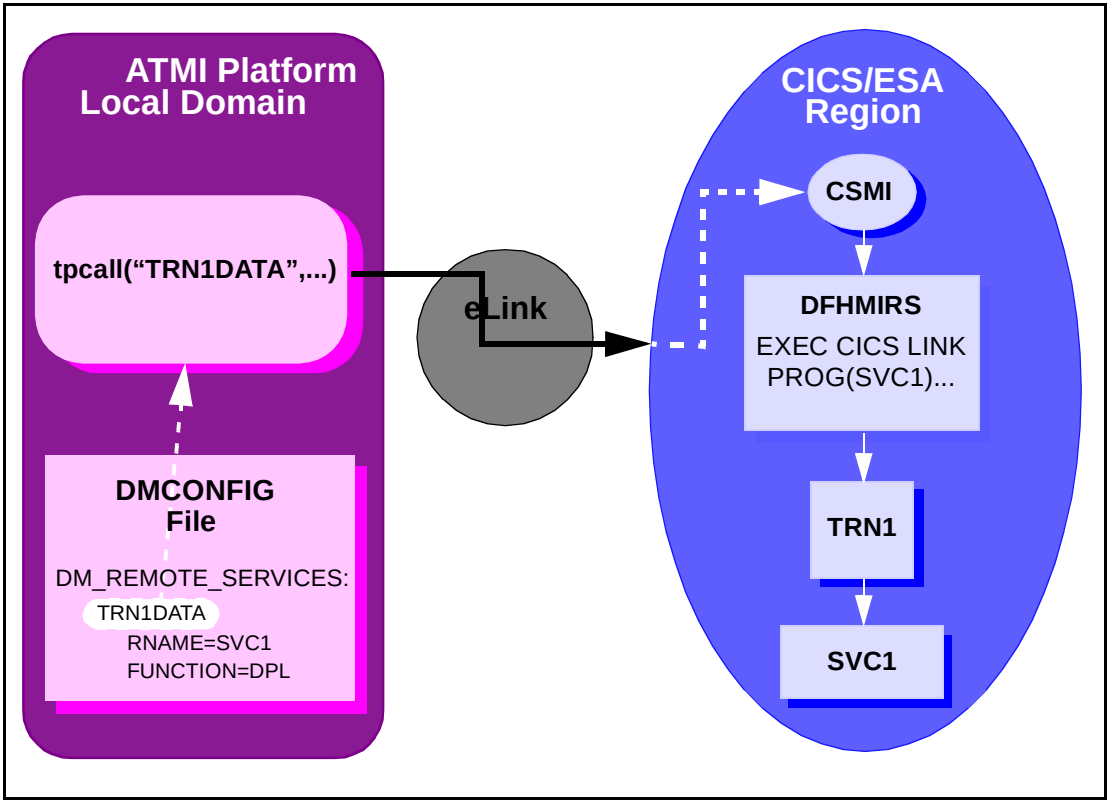


1. The CPI-C application program `MIRRCPIC` is invoked using environment start-up specifications.
2. The `MIRRCPIC` client requests `cminit` to establish conversation attributes and receive a conversation ID that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cmssl` sets the conversation attributes to sync-level 2 with `CM_SYNCPOINT`. This allows the ATMI service to participate in the transaction.
4. The `cmallc` request initiates the advertised service mapped to `MIRRORSERV` in the DM_LOCAL_SERVICES section of the DMCONFIG file.
5. The MIRRCPIC causes the client to relinquish control to the ATMI server with a prepare-to-receive request. The `cmsptr` sets the prepare-to-receive type to CM_RECEIVE_AND_FLUSH. The `cmptr` request immediately relinquishes control.
6. The `MIRROR` service sends the data contents of the `odata` buffer to the `cmrcv ibuffer`. The `cmrcv` receives a confirm request from the server indicating the conversation should be terminated.
7. The client replies positively to the confirm request with `cmcfmd`.
8. The `MIRRCPIC` client prepares to free the conversation with the `cmdeal` request. The conversation in CM_DEALLOCATE_SYNC_LEVEL commits all updated resources in the transaction and waits for the SAA resource recovery verb, `srrcmit`. After the `commit` sequence has completed, the conversation terminates.

CICS/ESA Mirror Transaction Examples

Implicit Attachment of TRANSID (Outbound Requests Only)

Figure 4-1 Implicit Attachment of TRANSID (Outbound Requests Only)

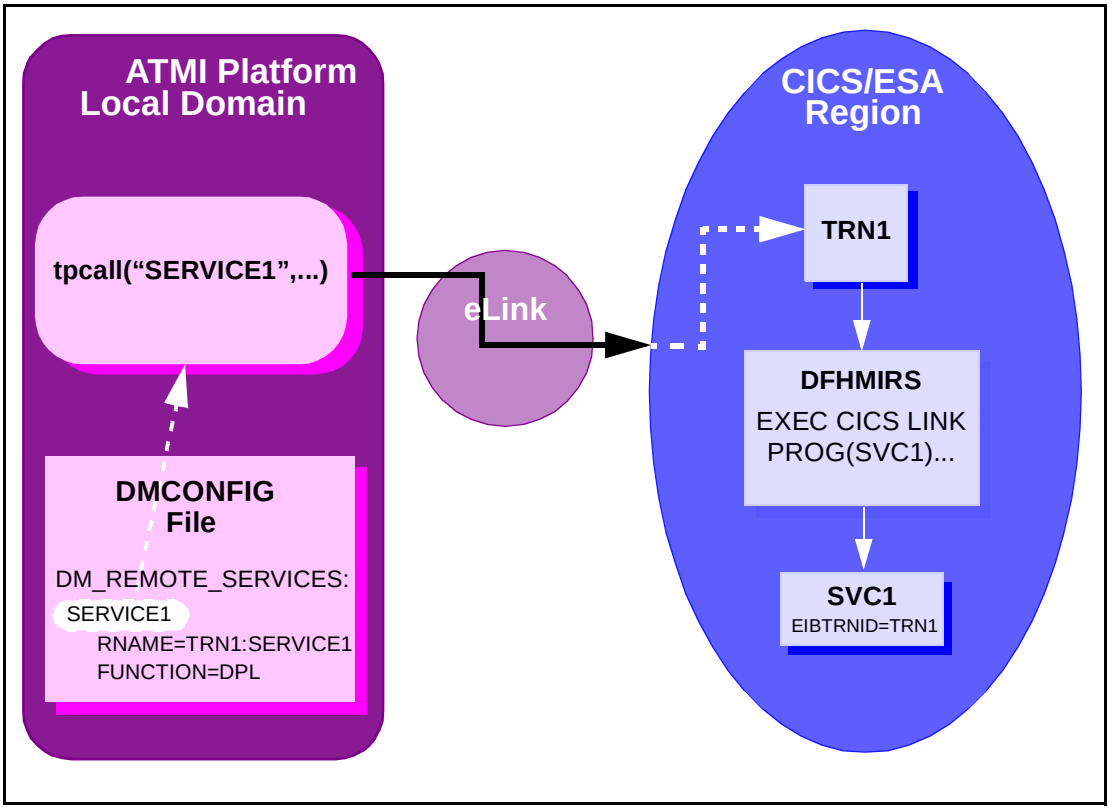


The following list describes the process for implicit attachment as illustrated in Figure 4-1:

1. The ATMI service makes a request to the service `TRN1DATA`, which is advertised as a remote service in the `DMCONFIG` file. It is a DPL request to a program named `SVC1` in the CICS/ESA region.
2. The first four characters of the remote service tag name (`TRN1`) are extracted and passed to the CICS/ESA region as the invoking `TRANSID`. No CICS/ESA resource definition for the `TRANSID` is required in the region.
3. The mirror transaction CSMI is attached in the CICS/ESA region, starting the mirror program `DFHMIRS`. The program performs the DTP requests for the service.
4. The mirror program now attaches the invoking `TRANSID` (`TRN1`) and then invokes the application service program `SVC1`.The program can interrogate the `EIBTRNID` field to find this value.

Explicit Attachment of TRANSID for Outbound Requests

Figure 4-2 Explicit Attachment of TRANSID for Outbound Requests

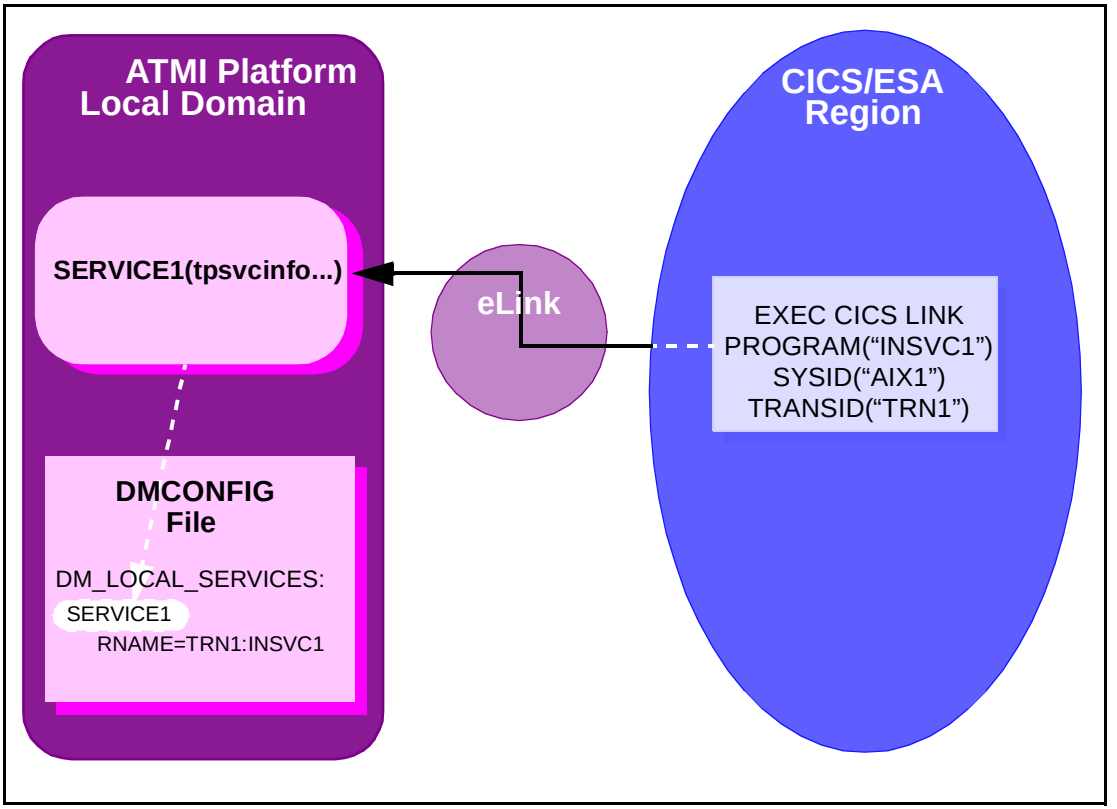


The following list describes the process for explicit attachment as illustrated in Figure 4-2:

1. The ATMI program makes a service request for `SERVICE1`, which is advertised as a remote service in the `DMCONFIG` file. The `FUNCTION` option indicates the remote service is invoked as a DPL.
2. The request extracts `TRN1` as an alternate mirror transaction ID for the remote region, along with the remote program name `SERVICE1`.
3. The `TRN1` ID is attached instead of the default mirror transaction, `CSMI` or `CVMI`. The `TRN1` ID must be defined as a transaction resource in the remote region and must point to the mirror transaction program `DFHMIRS`.
4. The mirror program `DFMMIRS` calls the server application program, passing the `TRN1` ID in the `EIBTRNID` field.

Explicit Attachment of TRANSID for Inbound Requests

Figure 4-3 Explicit Attachment of TRANSID for Inbound Requests



The following list describes the process for implicit attachment as illustrated in Figure 4-3:

1. The CICS/ESA program makes a request to `INSVC1`, which is a local ATMI service. The `SYSID` and `PROGRAM` values in the request identify the local system and the name of the local service. The `TRANSID` option indicates the mirror transaction to be initiated.

2. The `PROGRAM` and mirror `TRANSID` are extracted from the DPL request and are used to find an exact `RNAME` match in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.

3. The service `SERVICE1`, which is advertised locally in the ATMI platform application, is initiated.

Additional Information

Additional information on the subject of CICS/ESA Intersystem Communications may be found in the following IBM publications:

- *CICS/ESA Intercommunication Guide*, IBM publication No. SC33-0657
- *CICS/ESA Distributed Transaction Programming Guide*, IBM publication No. SC33-00783
- *CICS/ESA Recovery and Restart Guide*, IBM publication No. SC33-0658