Generic Attribute Profile

Functional Specification

RW-BLE-GATT-SW-FS

Version 11.01

2020-01-17

RivieraWaves
http://www.rivierawaves.com/

Revision History

Version	Date	Revision Description	Author
11.00	2019-06-18	First version with 5.2 features GATT part of Host document split	FBE / LT review
11.01	2020-01-17	Update versions to 5.2 number	FBE

Table of Contents

List of Figures

1 Overview

1.1 Document Overview

This document describes the embedded Software (SW) implementation of the Generic Attribute Profile (GATT) block as part of the RivieraWaves (RW) Bluetooth Low Energy (BLE) Stack. Its purpose is to explain architecture and behavior of this block and its interactions with other parts of RivieraWaves BLE IPs.
Even if this block is a proprietary block, this document requires the intended audience to have knowledge about the Bluetooth protocol stack. Please refer to the Bluetooth vMilan standard specification (see [1]).

1.2 Architecture Overview

As shown in Figure 1-1, GATT block is part of the BLE Host stack. GATT lies above the L2CAP and interfaces with higher layer protocols.
The Generic Attribute Profile (GATT) is the gateway in using the Attribute Protocol to discover, read, write and obtain indications of the attributes present in the server attribute, and to configure the broadcasting of attributes. The GATT lies above the Attribute Protocol and communicates with Generic Access Profile, higher layer profiles and application.

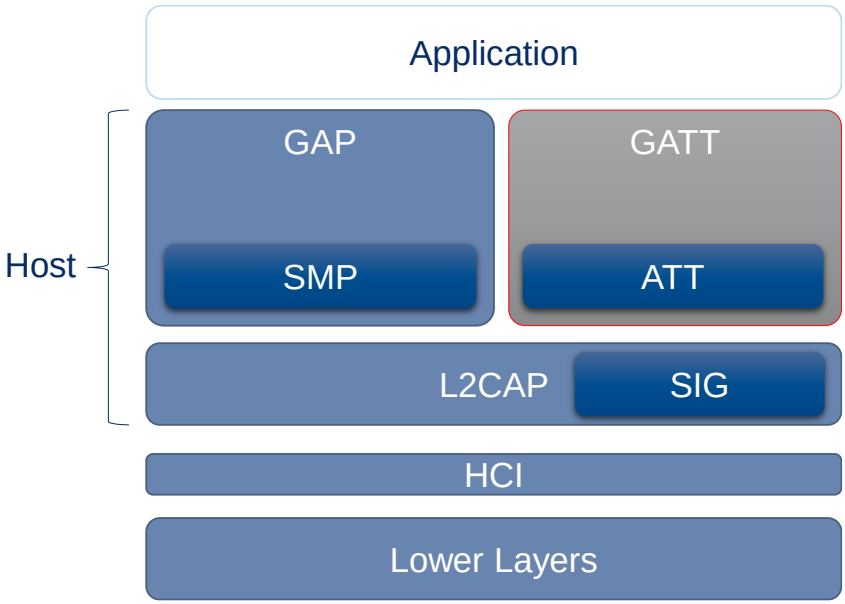

Figure 1-1: Position in the BLE Stack

In order to be able to exchange messages with upper layer interface, GATT block has its own task.

1.3 Presentation of GATT Sub-blocks

GATT module is composed by 6 sub-blocks (see Figure 1-2):

- User manager in charge of registered GATT users such as server and client profiles (see 2)
- Bearer manager in charge of attribute bearer usage and life cycle (see 4)
- Database manager which handle local services database (see 5)
- Procedure manager (see 6) that handle life cycle of:
  - Server procedures (see 6.1) which use attribute database information.
  - Client procedures (see 6.2).

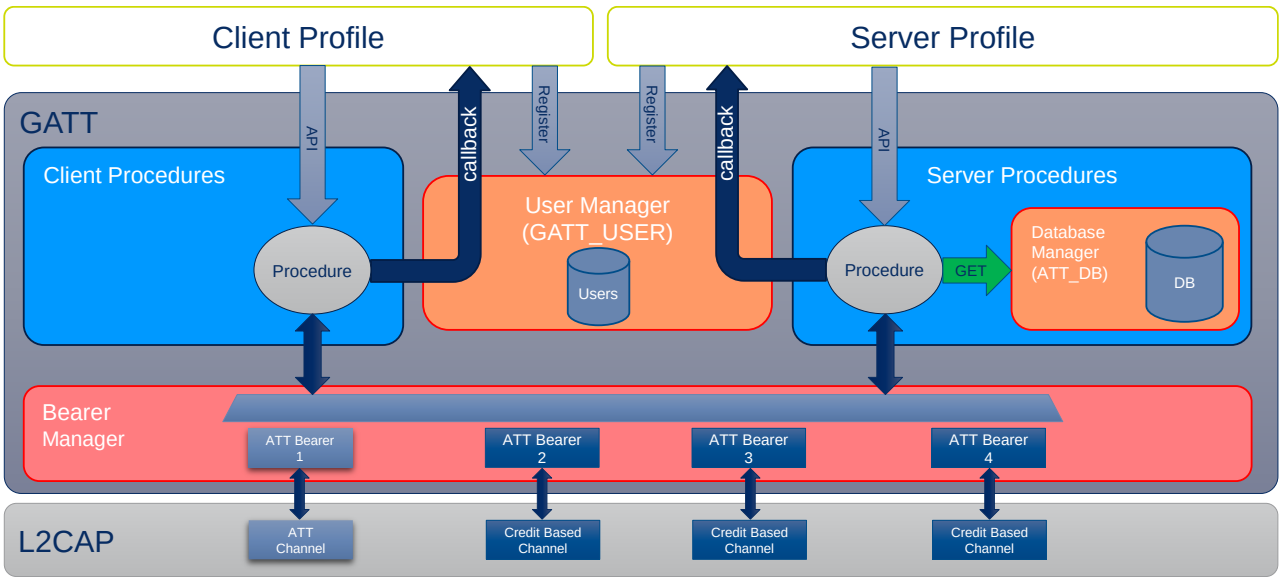

Figure 1-2: GATT Sub-blocks

2 GATT Users

In order to be able to use GATT client or server features, an upper layer module must first register itself as a GATT user. The registration can be done either by native or kernel message API. A GATT user can be either a profile server or a profile client but not both in same time.
Following information must be provided during registration:

- Task Identifier: task number that handles kernel messages API (unused for native API).
- Priority level: Used to prioritize usage of ATT bearers.
- Preferred MTU: This will be used to negotiate proper MTU value. Shall be greater than default MTU.
- Set of Callback functions: If native API is used, all functions defined in the set must be supported by the user, for kernel message API a default set is provided.

Once registration has been accepted, a GATT user local identifier (user_lid) is returned and must then be used for all GATT operations (see Figure 2-1).
The number of GATT users that can be registered is configurable at compilation and is at least equal to the number of profiles that can be added. It is recommended to register only one GATT user for a given profile.
Internally two GATT users are reserved for host stack.

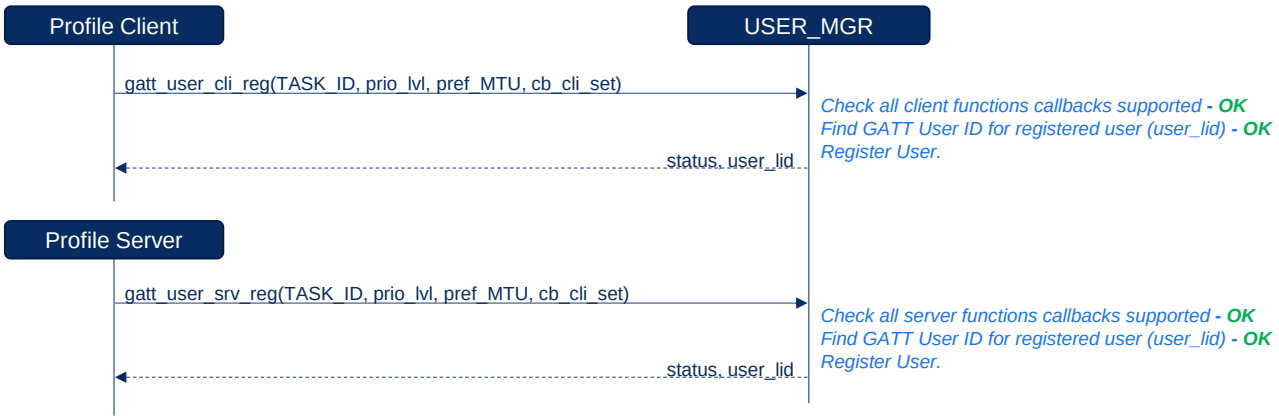

Figure 2-1: GATT User registration

Unregistering a GATT user is possible only if it has no associated service and no on-going procedure.
GATT and GAP layers are registered as GATT users during stack initialization.

3 Enhanced Attribute

The Enhanced Attribute (EATT) feature is used to create multiple ATT bearers using L2CAP COCs. It allows several client transactions to be processed in parallel. These ATT bearers can be established only when link is encrypted.
Before trying to autonomously establish new ATT bearers, GATT module must check if EATT Supported characteristic is present in peer device's database and claims support of the feature. To do so, GATT Service Discovery is started as soon as connection establishment is confirmed by upper layer application for an non-bonded peer device.
Once link is encrypted, and if peer device supports EATT, creation of new ATT bearers is initiated by GATT module according to number of GATT Client users registered and maximum number of ATT bearers per connections.
MTU is chosen according to GATT Users preferred MTU.
If new GATT Users are added during connection with greater MTU values, the ATT bearers MTU are reconfigured.

4 Bearer Manager

GATT uses L2CAP API and callback functions for transmission and receptions of attribute PDUs.
When a new Attribute bearer is created, an environment structure is allocated to manage a list of reception buffers (see Figure 4-1), two transaction timers and tokens for on-going procedures (see 6).

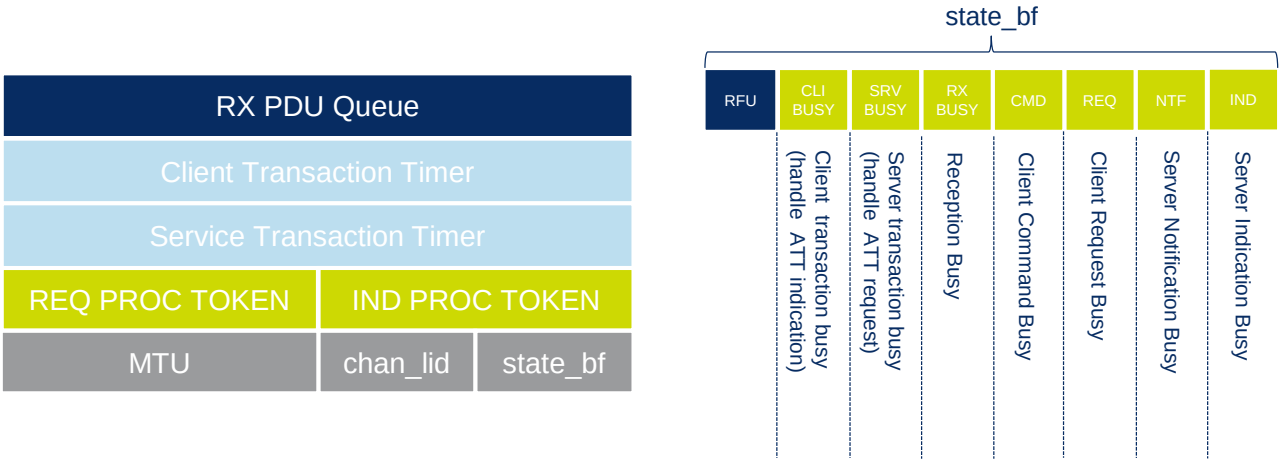

Figure 4-1: Bearer Manager environment structure.

When an ATT PDU is received, it's put into proper ATT Bearer queue so that it can be process as soon as possible. If there is nothing in process on received bearer (see Figure 4-2), this can be done immediately.

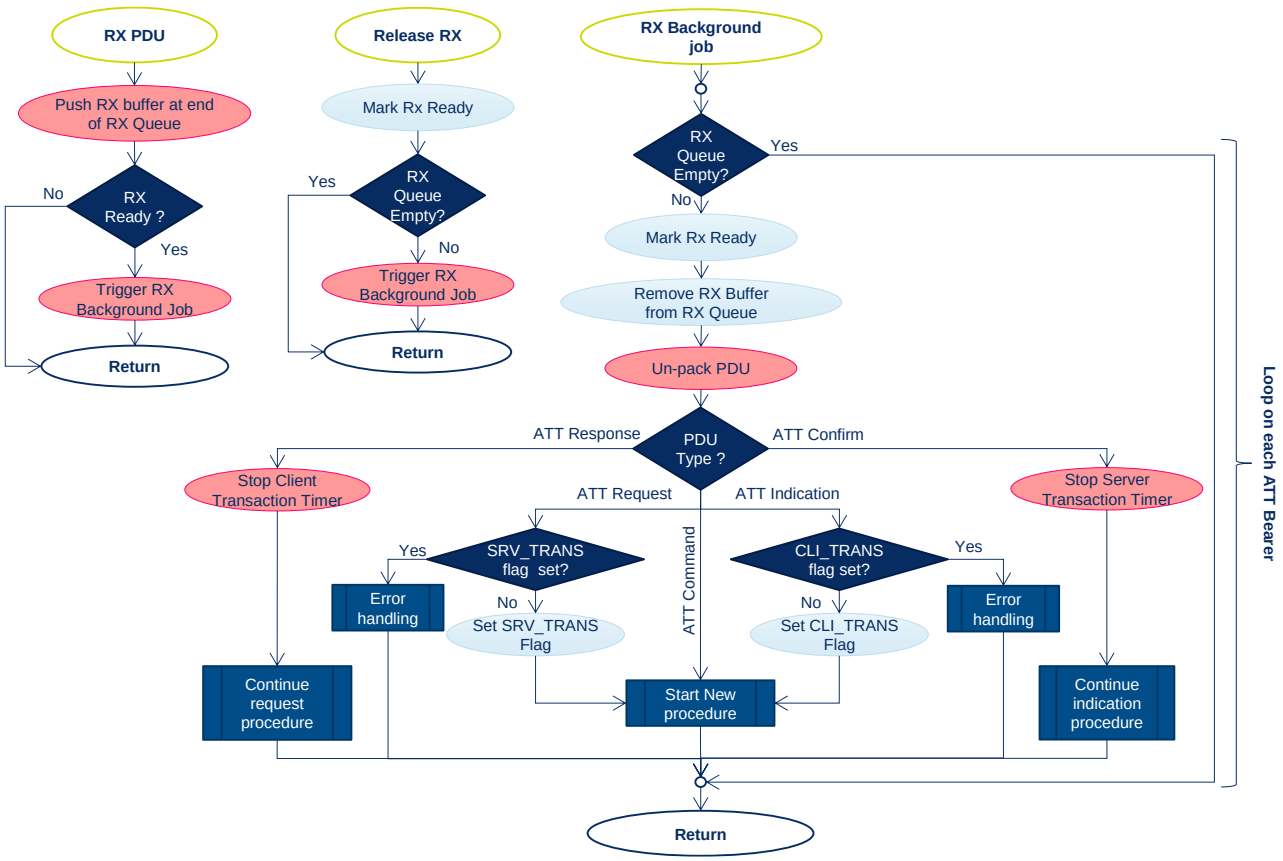

Figure 4-2: ATT PDU reception and un-packing state machine

When an ATT PDU is sent, transaction timer is started for requests and indications. Corresponding attribute flag is set in order In order to prevent Procedure manager from starting a new procedure that would use the same attribute type (see Figure 4-3).
When status about a done transmission is received from L2CAP module, the attribute flag is cleared and corresponding procedure is resumed.

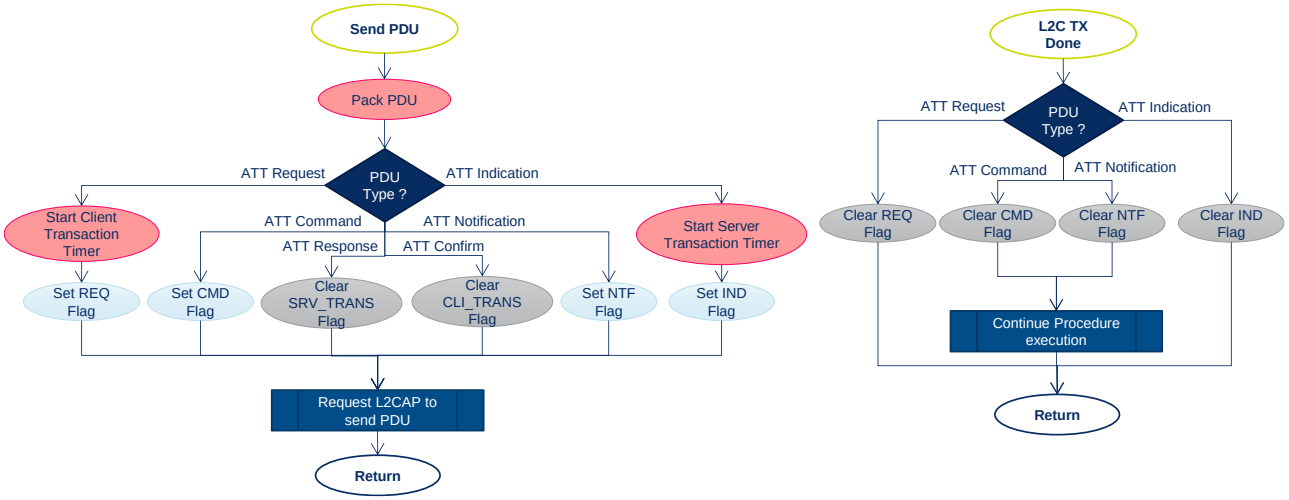

Figure 4-3: ATT PDU transmission state machine

5 Database Manager

Attribute Database (ATT_DB) module manages the attribute database as a list of service description structures sorted by ascending order of service start handle. These structures are dynamically allocated from the Attribute Heap.
In order to allocate a new service, database manager provides an API function. A service start handle can be chosen by GATT user. If not set, a start handle is autonomously chosen.

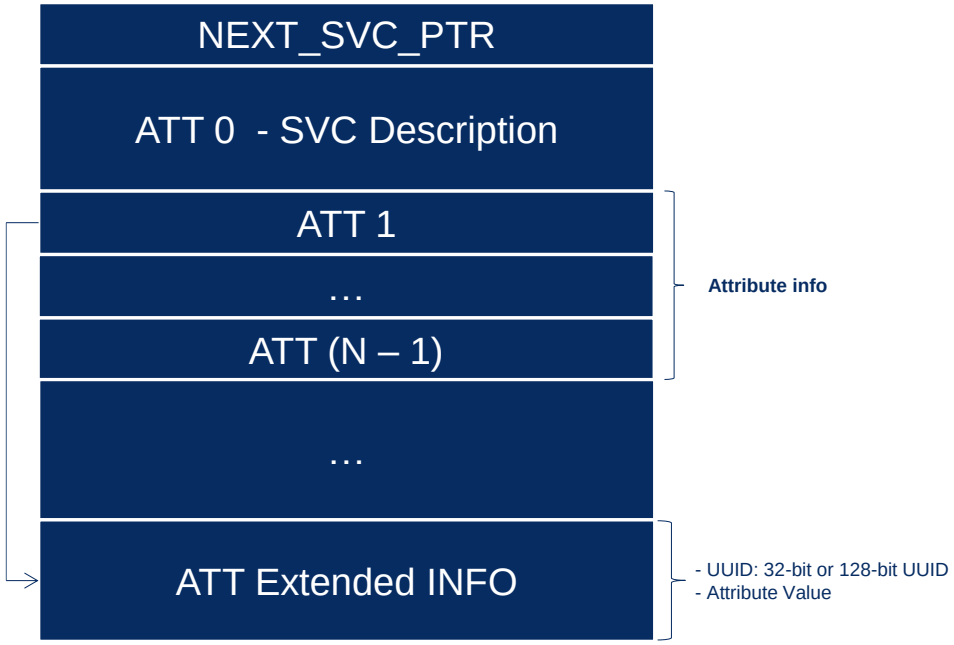

Figure 5-1: Service Description block of ATT database.

Figure 5-1 exposes content of service description structure. This structure has a pointer to next service (NEXT_SVC_PTR), its start handle and the last handle value. It also has an array of attributes definition (see Figure 5-1).
First attribute into service describes the service (see Figure 5-2). It is used to know services permission and number of attributes present into the service. It's forbidden to have several services attribute into a service structure.
Finally end of memory block is used to retrieve 32-bits or 128-bits UUIDs and attribute values that can be read from the database.
Note: Attribute handle are unique, services handles have to be exclusive.
Note: Services handles mapping should be fixed in order to prevent collector to perform discovery at each connection.

5.1 Service Definition

A service is described with a 8 bytes field:

- GATT User local Identifier (user_lid)
- Service Start and Service End Handles
- Service permissions
- Service UUID

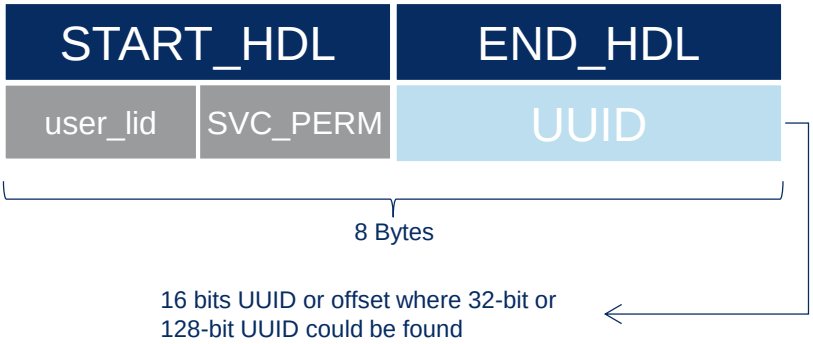

Figure 5-2: Service descriptor.

Note: If Service UUID is a 32-bit or 128-bit UUID, UUID value contains an offset (from beginning of service description structure address) allowing to retrieve the complete UUID value.

Figure 5-3 illustrate service permission bit field:

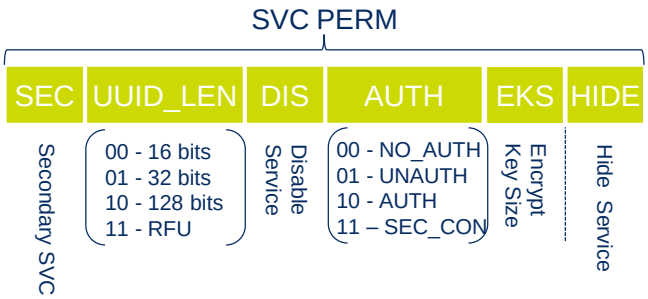

Figure 5-3: Service Permission field

- **SEC**: Used to know if service is a secondary or a primary service.
- **UUID_LEN**: Provide Service UUID length (16, 32 or 128 bits). If length is 32 or 128 bits, the UUID field contains offset pointer.
- **DIS:** is used to disable a service, it is visible by peer device but automatically rejects any attribute requests.
- **AUTH**: Force a level of authentication for attributes composing the service. This has no impact on attributes which are Read-Only mandatory.
- **EKS**: Use of a 16-bit encryption key is required for attribute requiring an authentication level.
- **HIDE**: Hide service so that it cannot be used by peer devices without removing it from database.

5.2 Attribute Definition

An attribute is a 6 bytes field used to describe its UUID, its permissions and some extended information such as:

- Pointed handle
- Maximum Attribute Length
- Value (for some specific attribute types)

Note: if Attribute UUID is a 32 or 128 bits UUID, UUID value contains offset in the service block where it can be found.
Note: describes Attribute types specified by Core Specification (see [1])

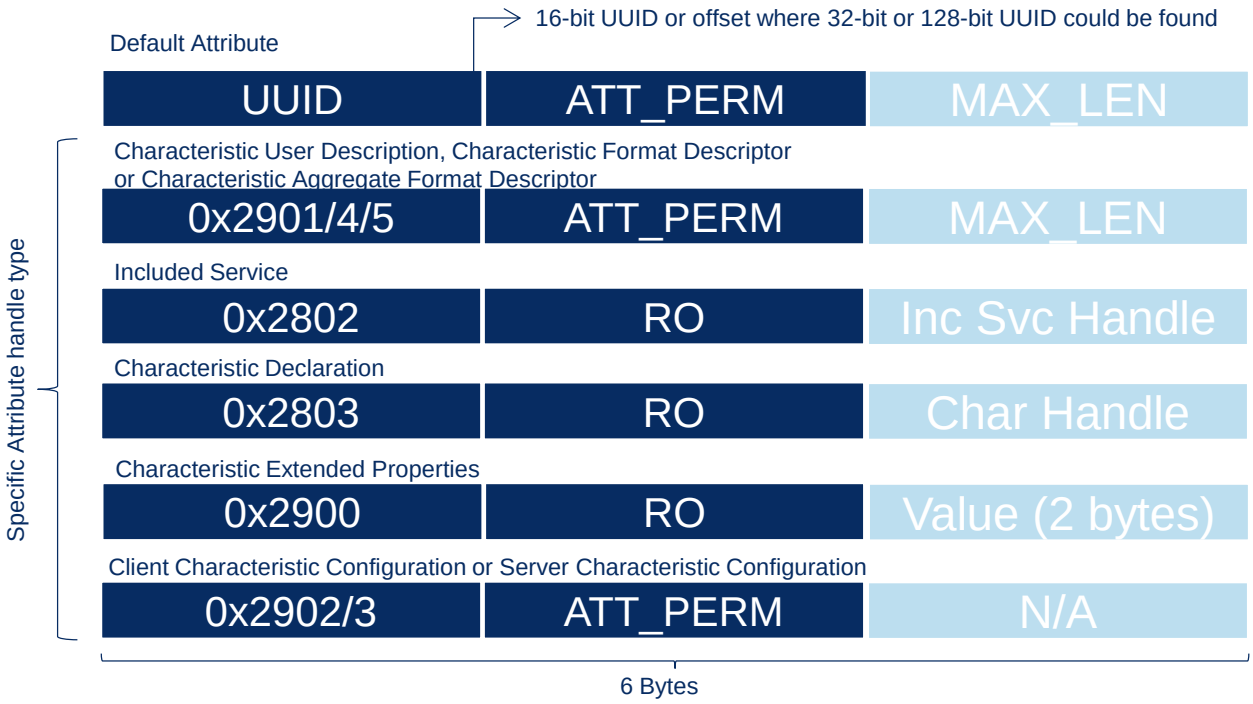

Figure 5-4: Attributes types.

Figure 5-5 illustrates attribute permission bit field:

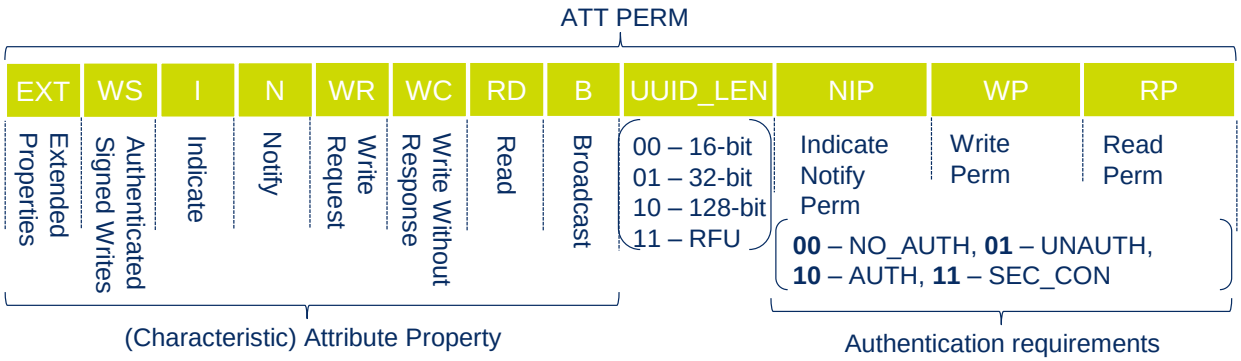

Figure 5-5: Attribute Permission field

Following field used to generate Characteristic declaration properties value:

- **RD:** Read attribute allowed
- **WR:** Write Request allowed on current attribute
- **WS:** Write Signed allowed on current attribute
- **WC:** Write without response allowed on current attribute
- **N**: Notification event allowed
- **I**: Indication event allowed
- **B**: Attribute value can be broadcasted using advertising data *(SCC descriptor shall follow)*
- **EXT**: Extended property field present *(CEP descriptor shall follow)*

Attribute Authentication requirements

- **WP:** Write permission allowed with a certain level of authentication
- **RP:** Read permission allowed with a certain level of authentication
- **NIP**: Notification/Indication allowed with a specific level of authentication *(CCC descriptor shall follow)*

Note: For an attribute value, permissions are used to generate Characteristic Description property value.

Other Attribute information

- **UUID_LEN**: Attribute UUID length (16, 32 or 128 bits). If length is 32 or 128 bits, the UUID field contains offset pointer.

5.3 Example

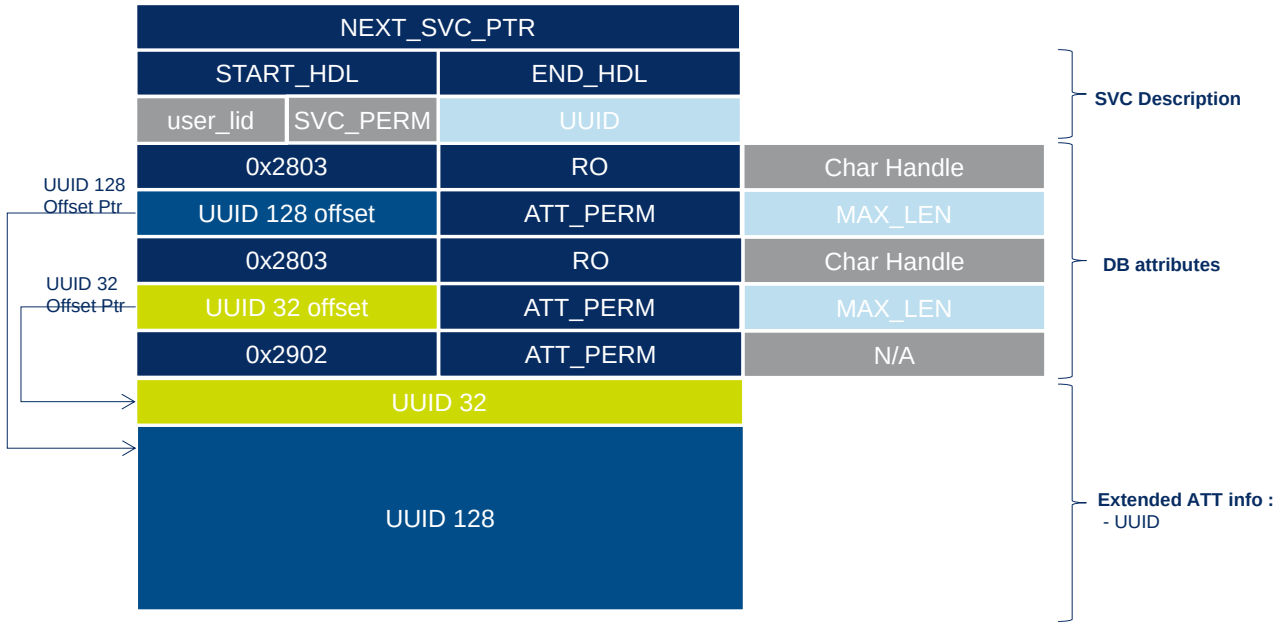

Figure 5-6: Attribute database example

6 Procedure Manager

Procedures are created by GATT user when it wants to access peer database (read, write, and discovery) or by a peer device upon reception of an attribute request, command, notification or indication.
The procedure can be either a client or a server procedure. It is composed of an operation environment, operation function, and a state machine.
In initialization phase, procedure environment must be allocated using procedure manager. This action assigns a token number to the created procedure. This token is used during all procedure action and helps procedure manager to retrieve procedure environment.
If initiated by GATT user, the procedure must be associated to an attribute bearer before executing it. If no bearer is available, the procedure is put into a wait state until a bearer can grant it (see 4).
A procedure can interact with a GATT user using registered callback functions (see 2).

Figure 6-1 and Figure 6-2 MSCs illustrates GATT user initiated procedures.

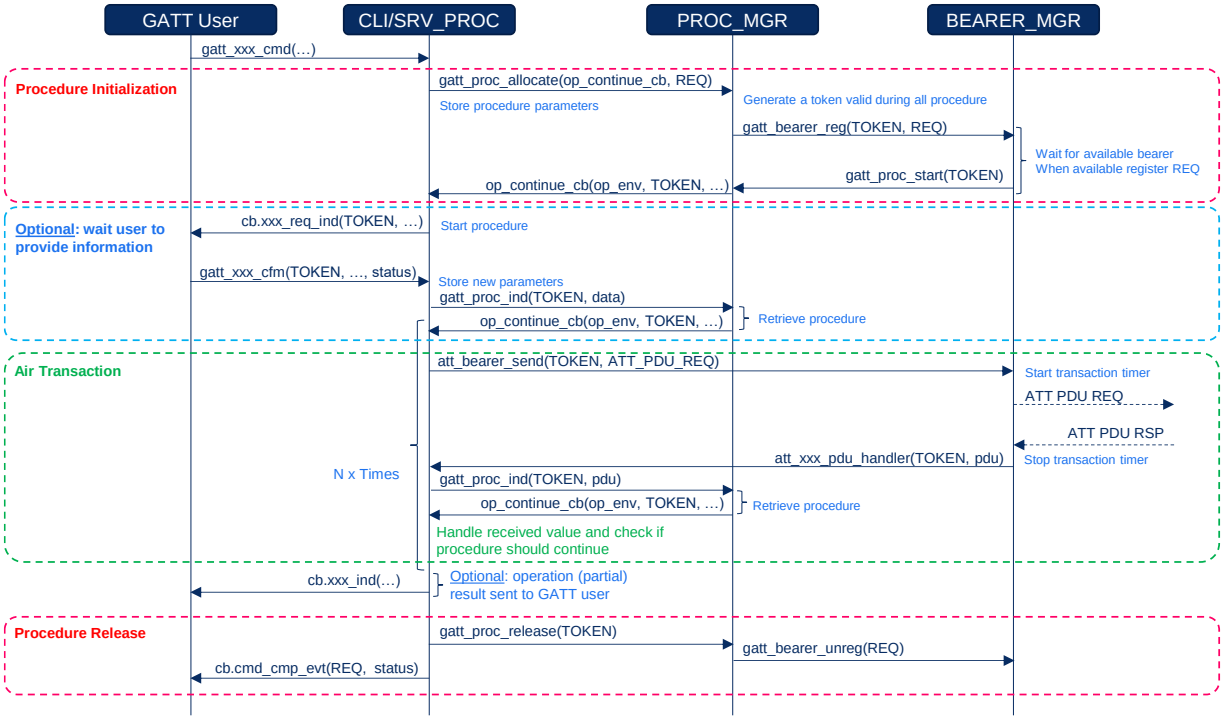

Figure 6-1: Procedure initiated by GATT user using ATT transaction (request or indication)

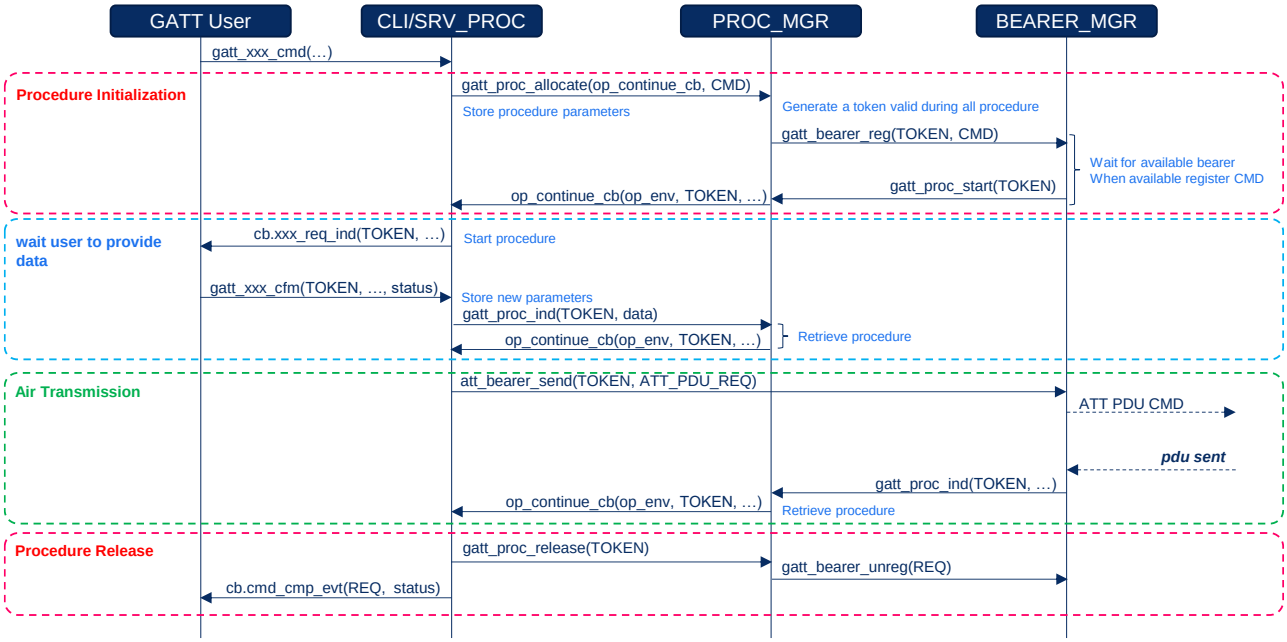

Figure 6-2: Procedure initiated by GATT user using ATT command or notification

When a server procedure is initiated by peer device, the GATT user identifier is retrieved according to targeted attribute handle (see 5).
When a client procedure is initiated by peer device, GATT user identifier is retrieved according to client registered handle (see 6.2.1).
Figure 6-3 and Figure 6-4 MSCs illustrate procedure initiated by peer device for server ATT PDU handling or client indication or notification reception.

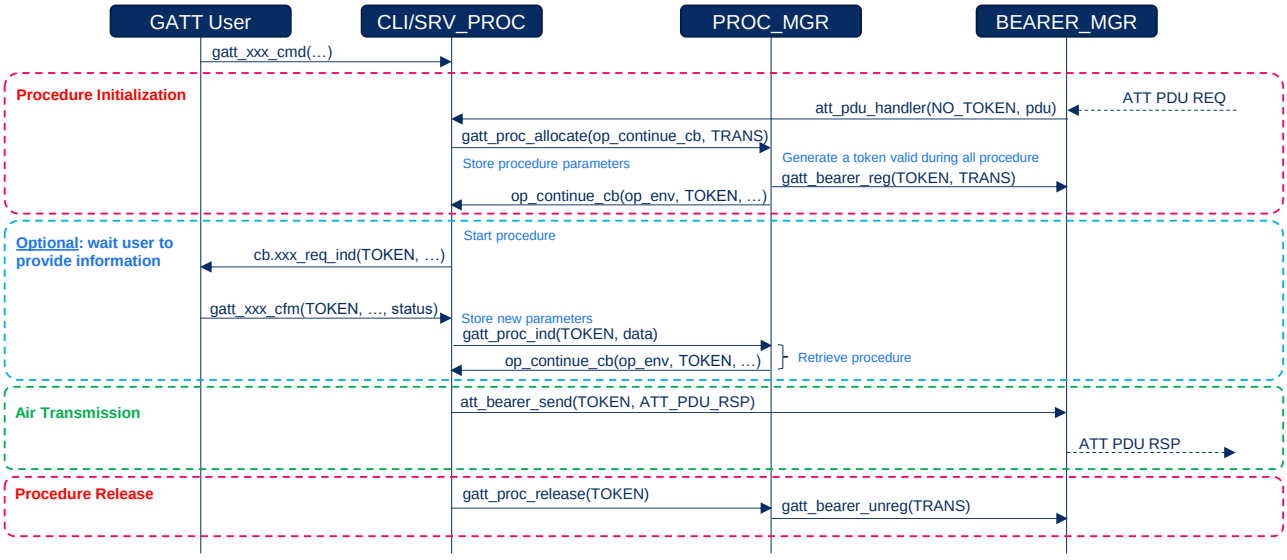

Figure 6-3: ATT transaction initiated by peer device

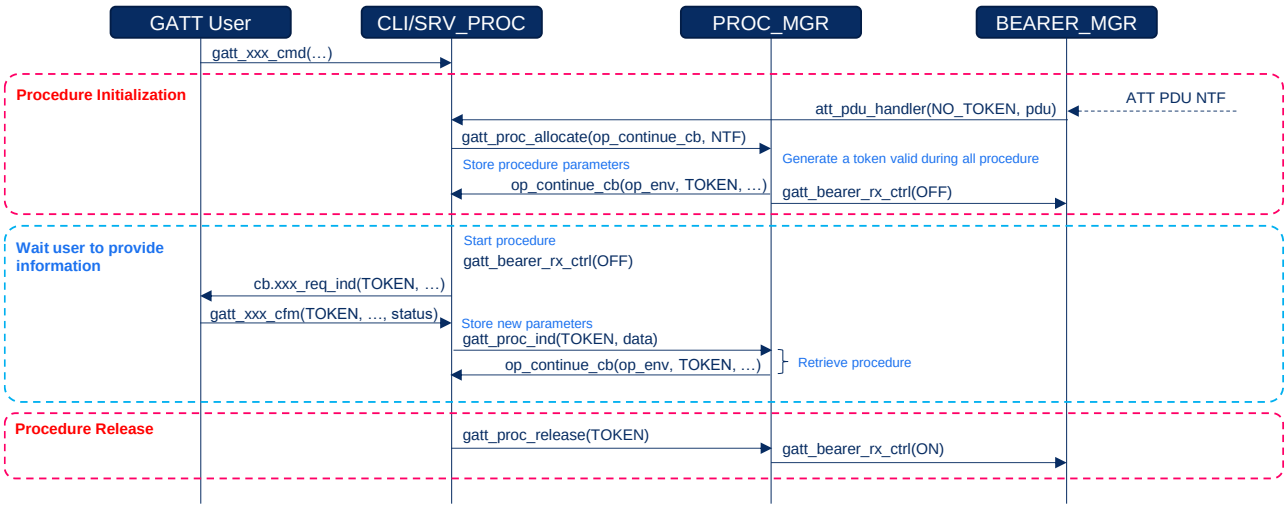

Figure 6-4: ATT notification or command received from peer device

6.1 Server Procedure

Attribute server have direct (function call) interface with Attribute database. It uses this interface to browse services and read characteristics values.

6.1.1 Attribute Discovery / Read

Attribute discovery procedures and reading procedures must take care of the ATT bearer MTU.
Also attribute value for an attribute read can be available after a background procedure execution (Profile cannot provide value immediately and must wait for application interaction), so the procedure must be paused until value is available.
During a discovery or read procedure, a list of transport buffer cache that contains partial response value is used. When procedure response can be sent, buffer cache is used to copy extracted data into a single transport buffer.

Search algorithm is described in Figure 6-5.

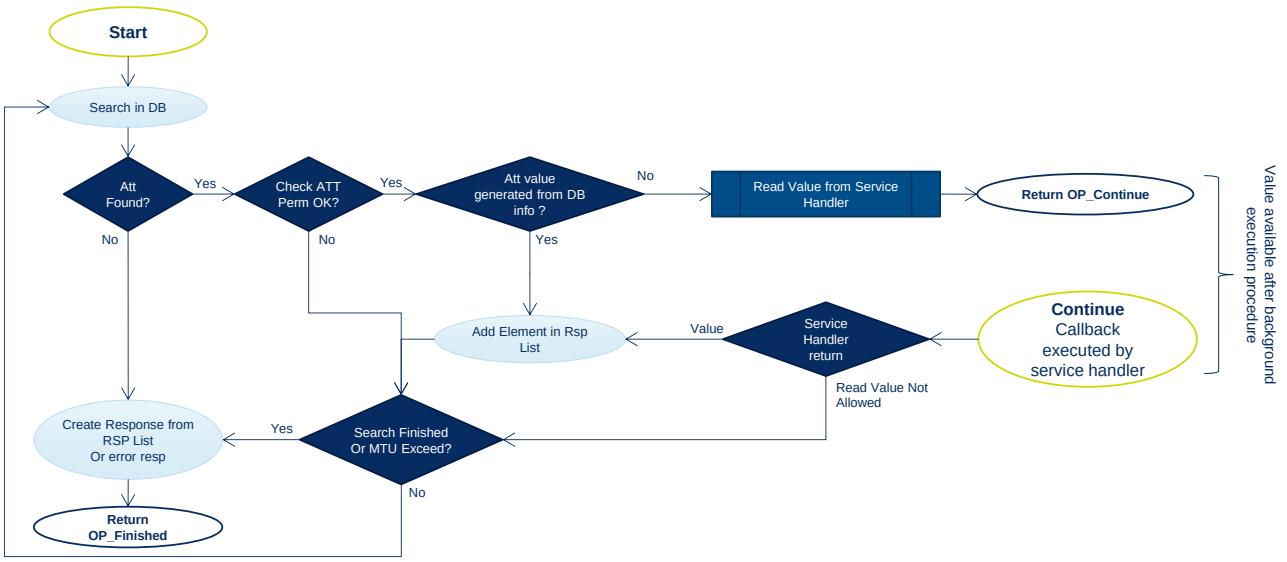

Figure 6-5: Attribute discovery state machine

6.1.2 Attribute Write

Following figure describes different type of write procedure in ATT

- **Write Command**
- **Write Request**
- **Write Signed**

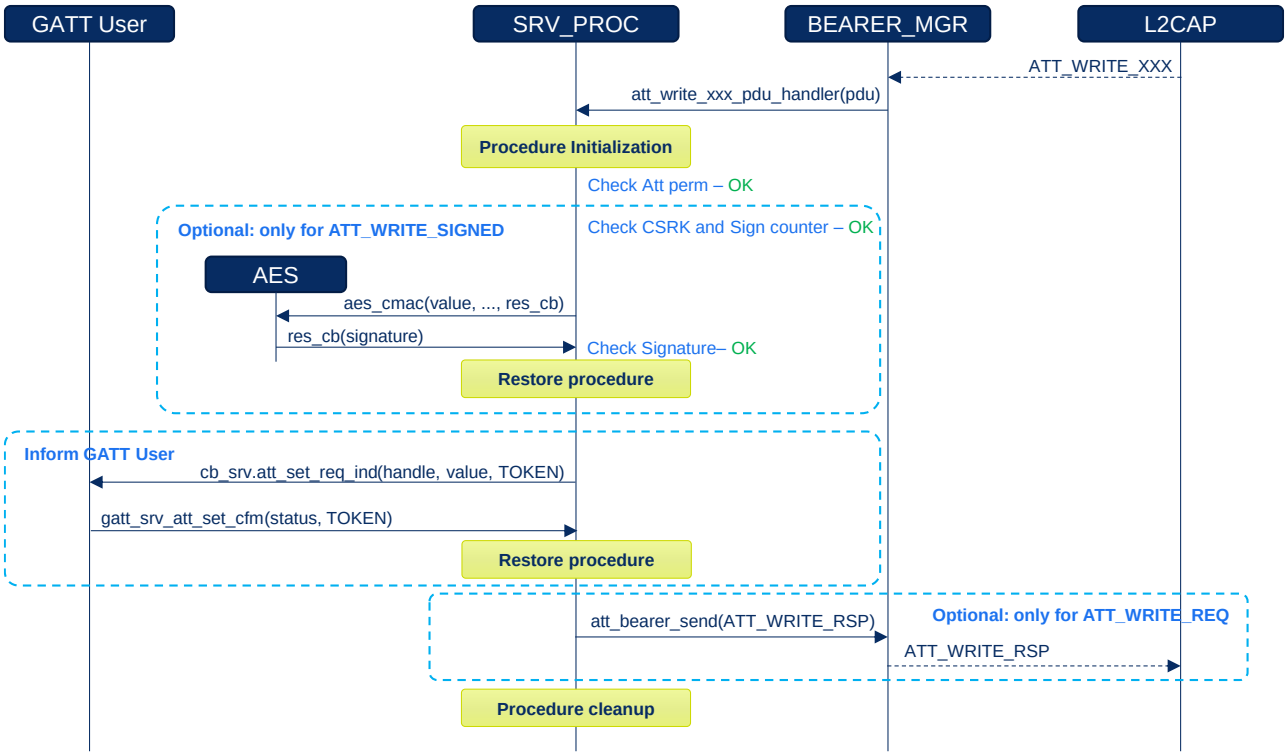

Figure 6-6: Write Command and Request MSC

- **Write Long/Multiple:** to reduce the number of copy, the transport buffer is reused for the prepare write response and be kept in prepare write lost.

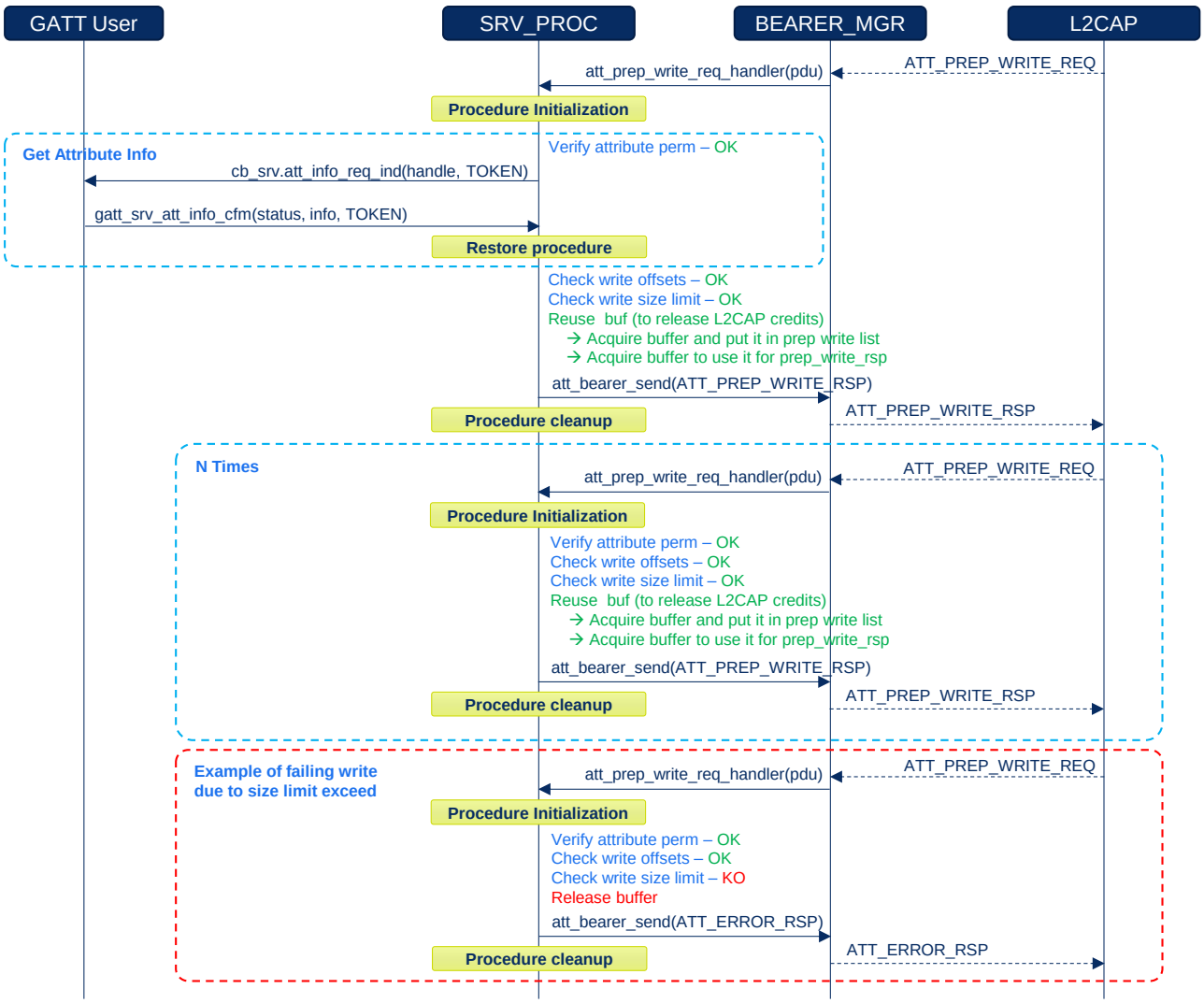

Figure 6-7: Multiple prepare write MSC

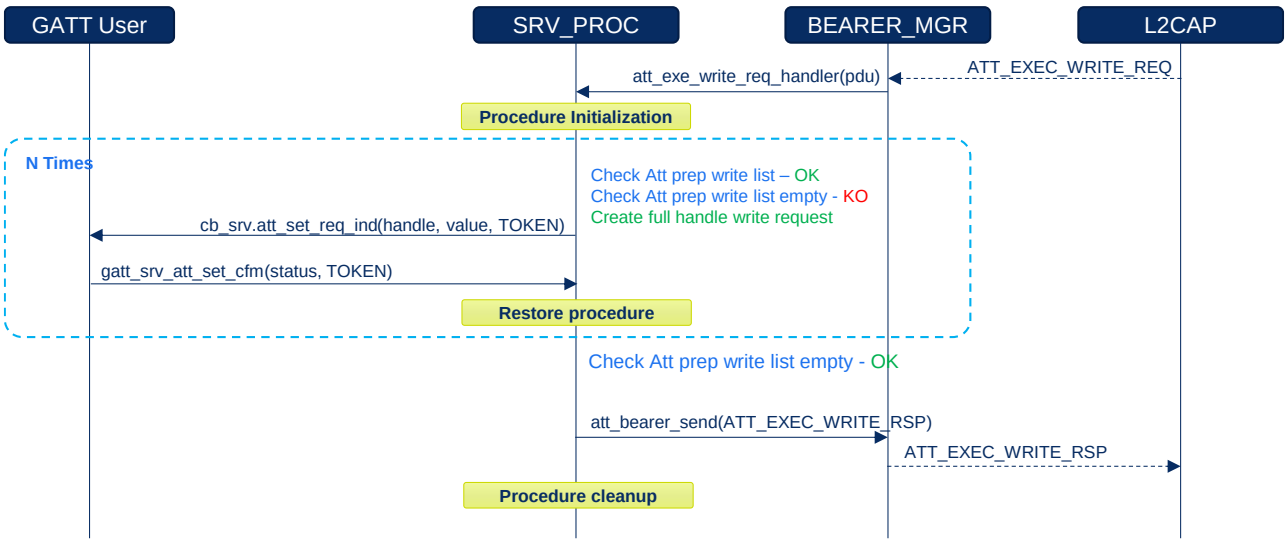

Figure 6-8: Execute write MSC

Note: for a proper flow control handling, all write request must be confirmed by profile.

6.1.3 Data Caching

Prepare write cache:

For non-atomic write, a cache is required. This cache is feed by prepare write and flushed during execute write request. This cache is part of attribute server environment since there is only one prepare queue for all ATT Bearers. Once all ATT bearer are closed, prepare write queue shall be purged.

6.1.4 Server Initiated events

Attribute server can be used to trigger some indication or notifications:

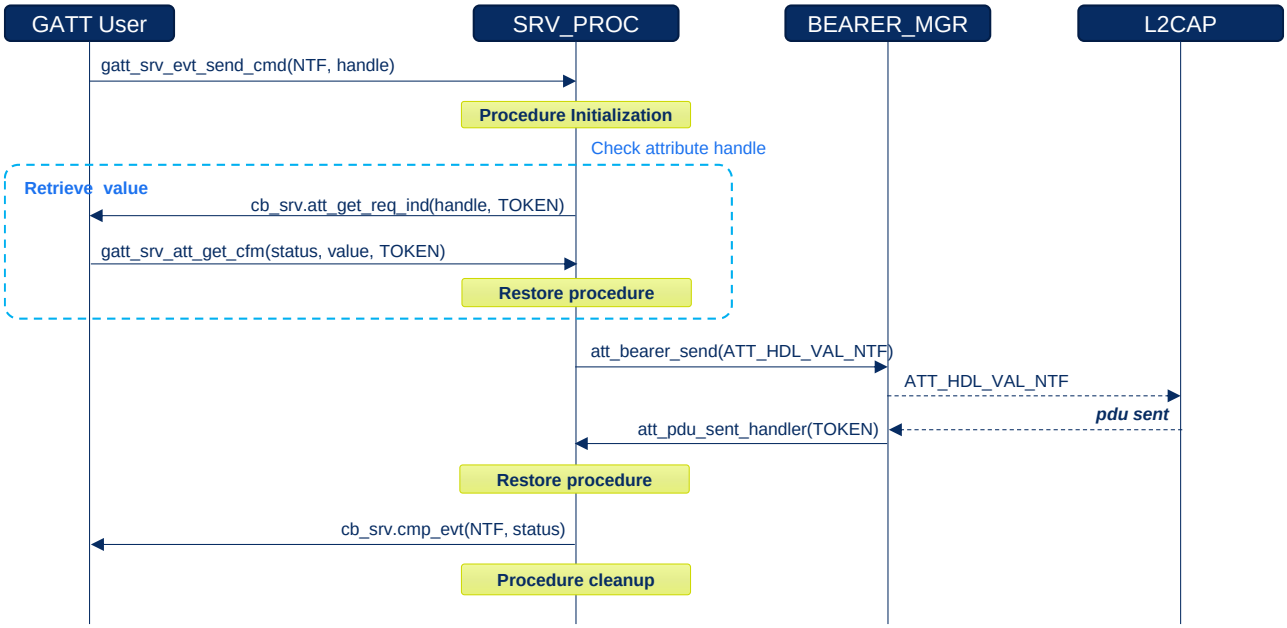

Figure 6-9: Trigger notification MSC

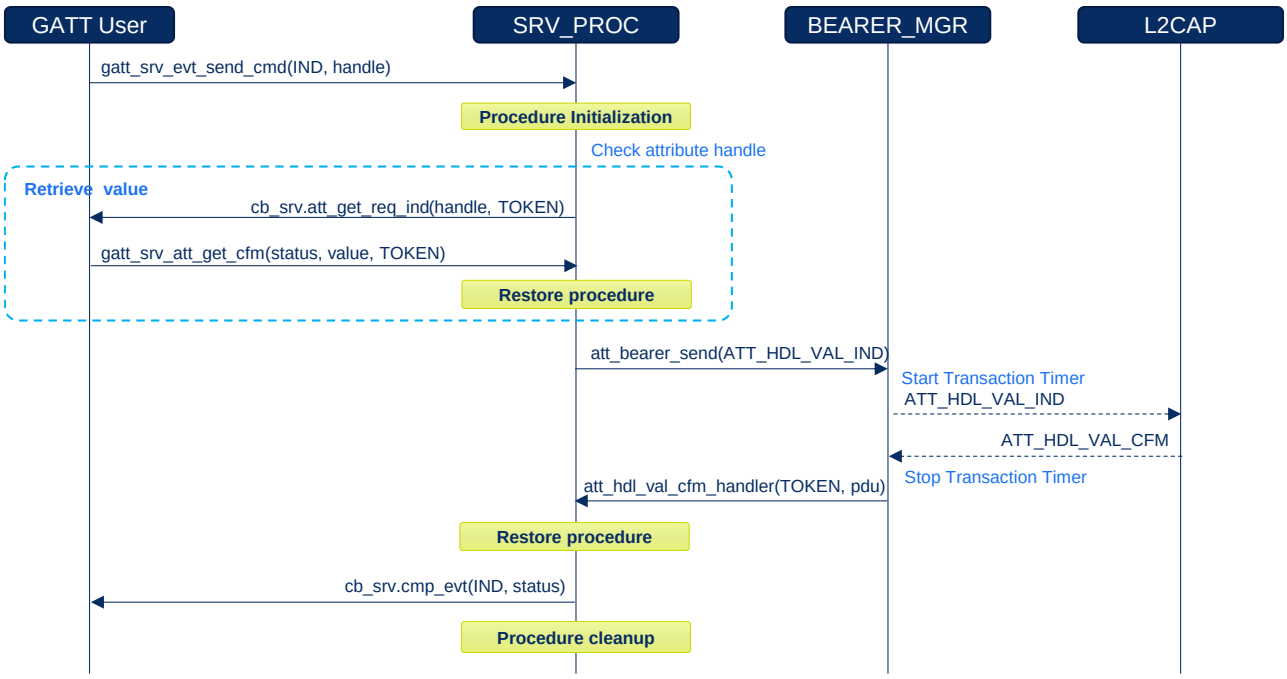

Figure 6-10: Trigger indication MSC

Note: Notification/Indication data is present into the event message. This event message can be used to update database value (If attribute value present in database).

6.2 Client Procedure

Attribute client conveys notification and indication to the registered client.

6.2.1 Reception of Notification or Indications

A GATT user can receive notifications or indications from a peer device if it has registered to service events. This can be done by providing peer service handle range (see Figure 6-11).

Note: By default application is informed of any received events if no registration has been performed.

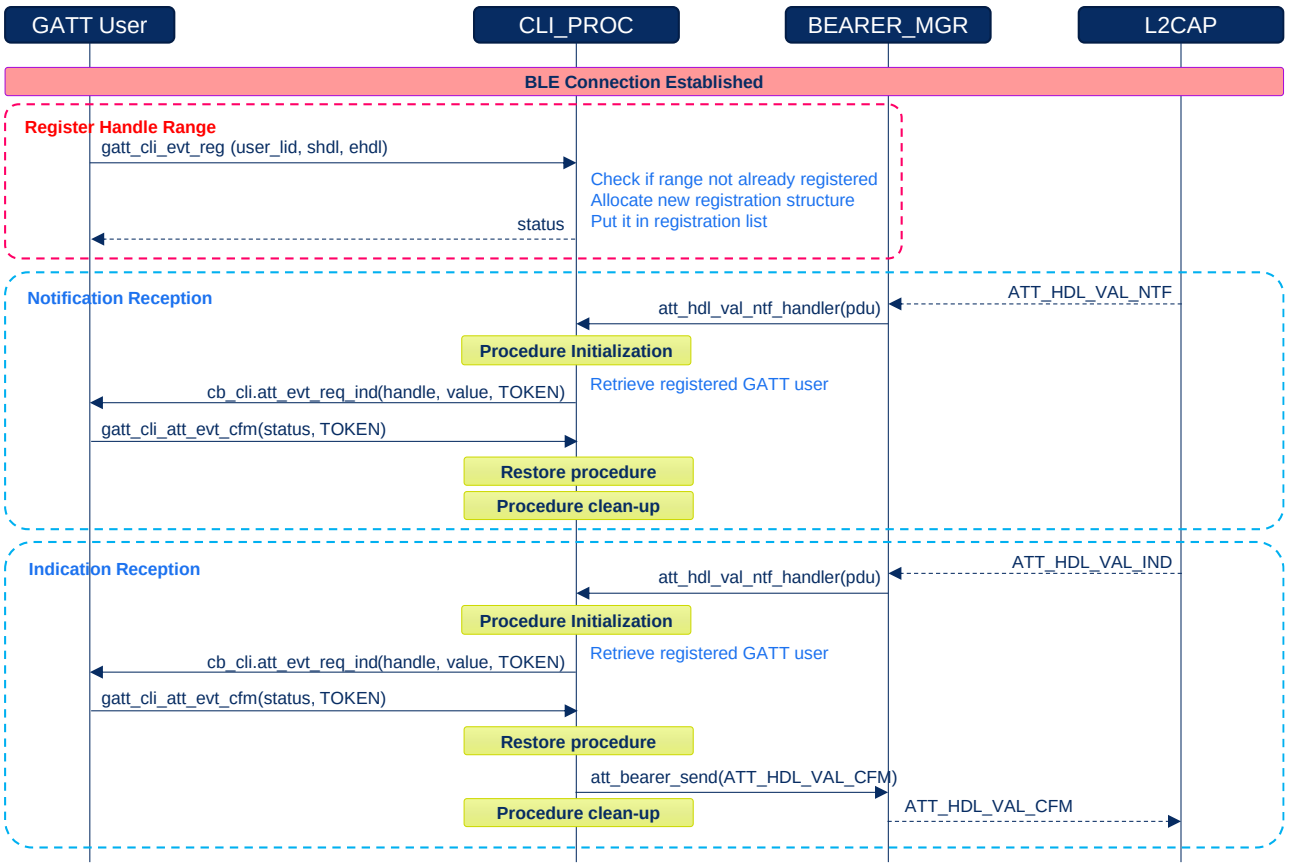

Figure 6-11: Client handle registration, reception of notification or indication from peer device MSC

6.2.2 Discovery Command

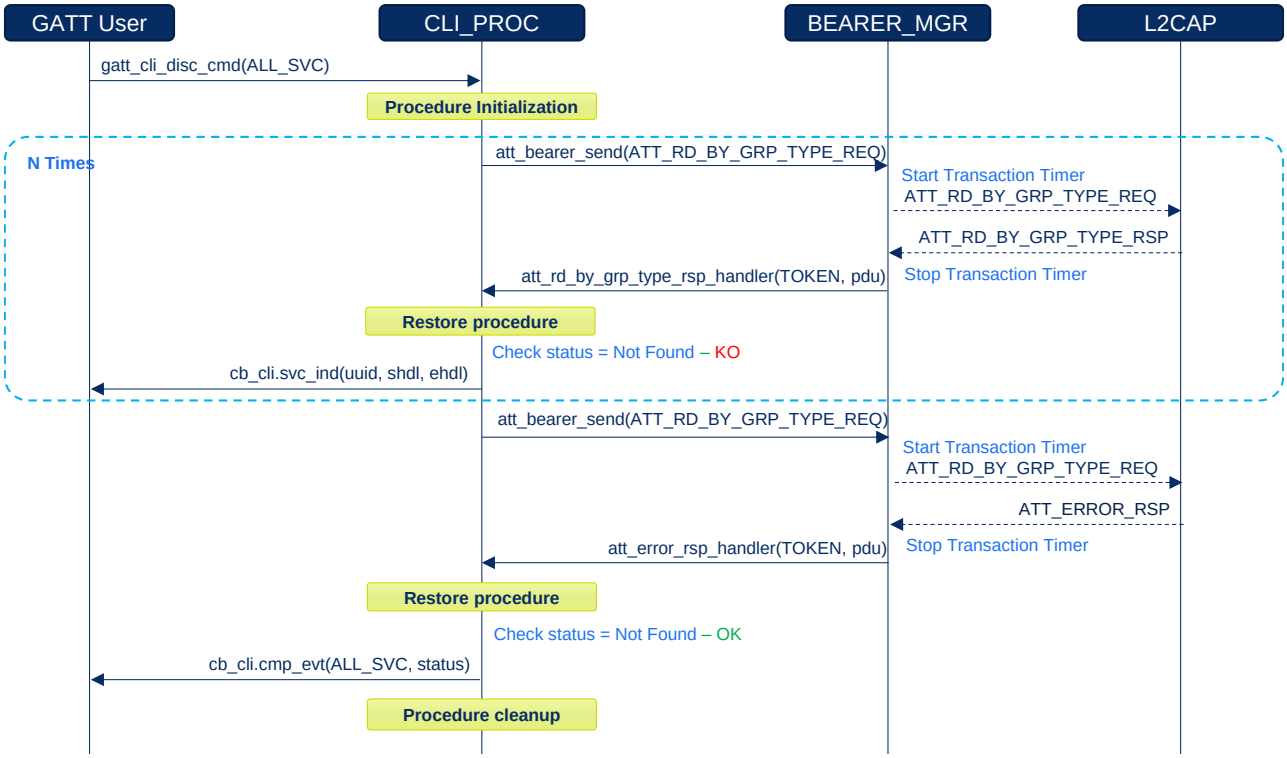

Figure 6-12: Discover all peer services MSC

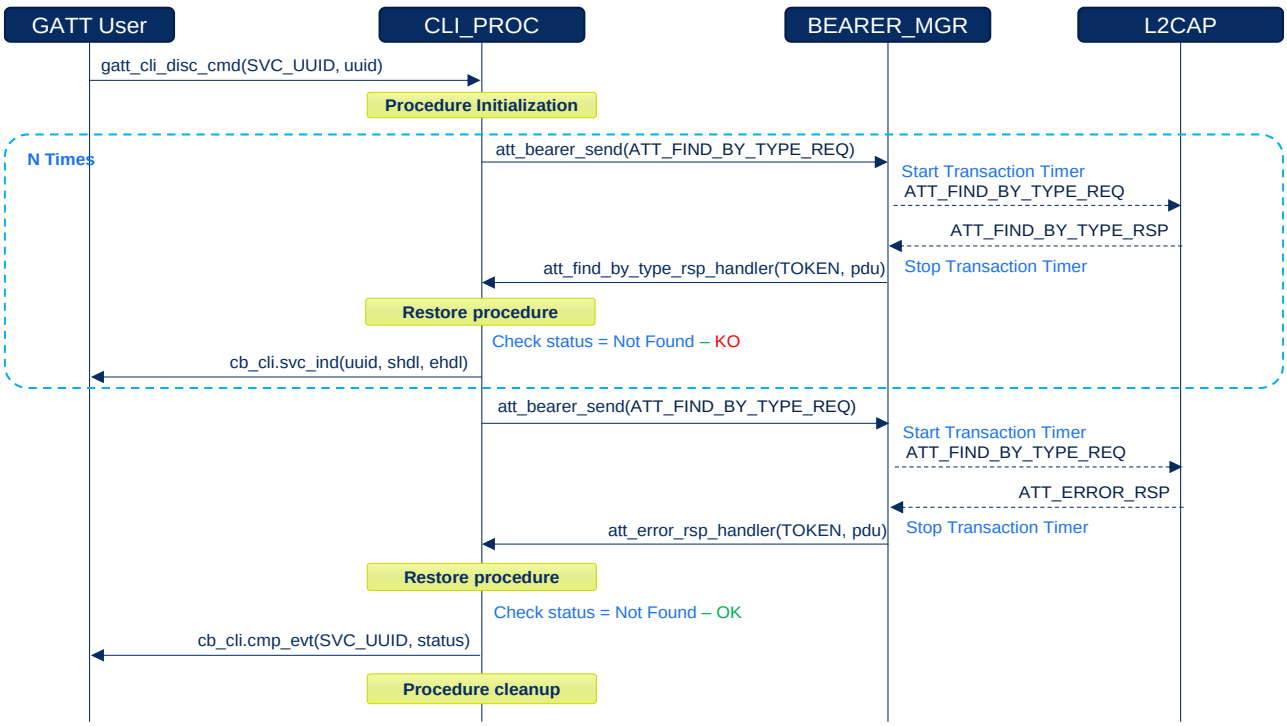

Figure 6-13: Discover peer services with specific UUID MSC

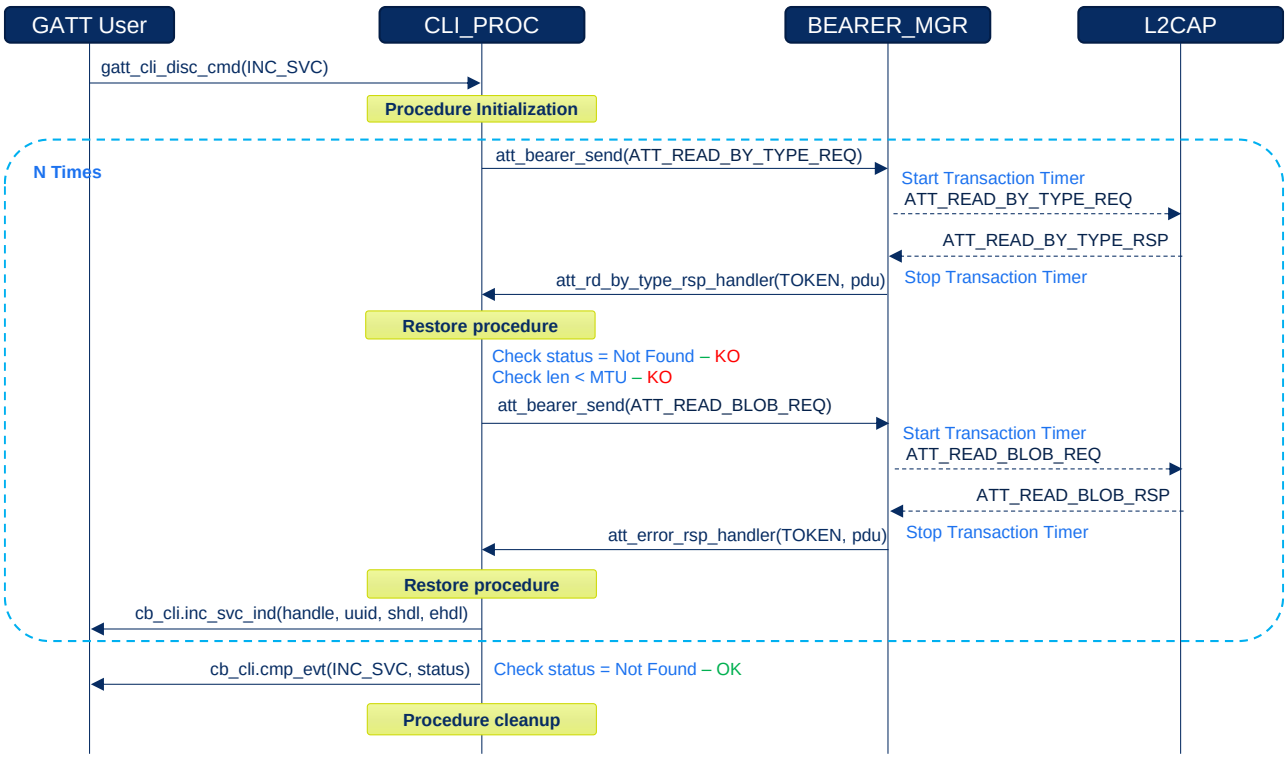

Figure 6-14: Discover peer included services MSC

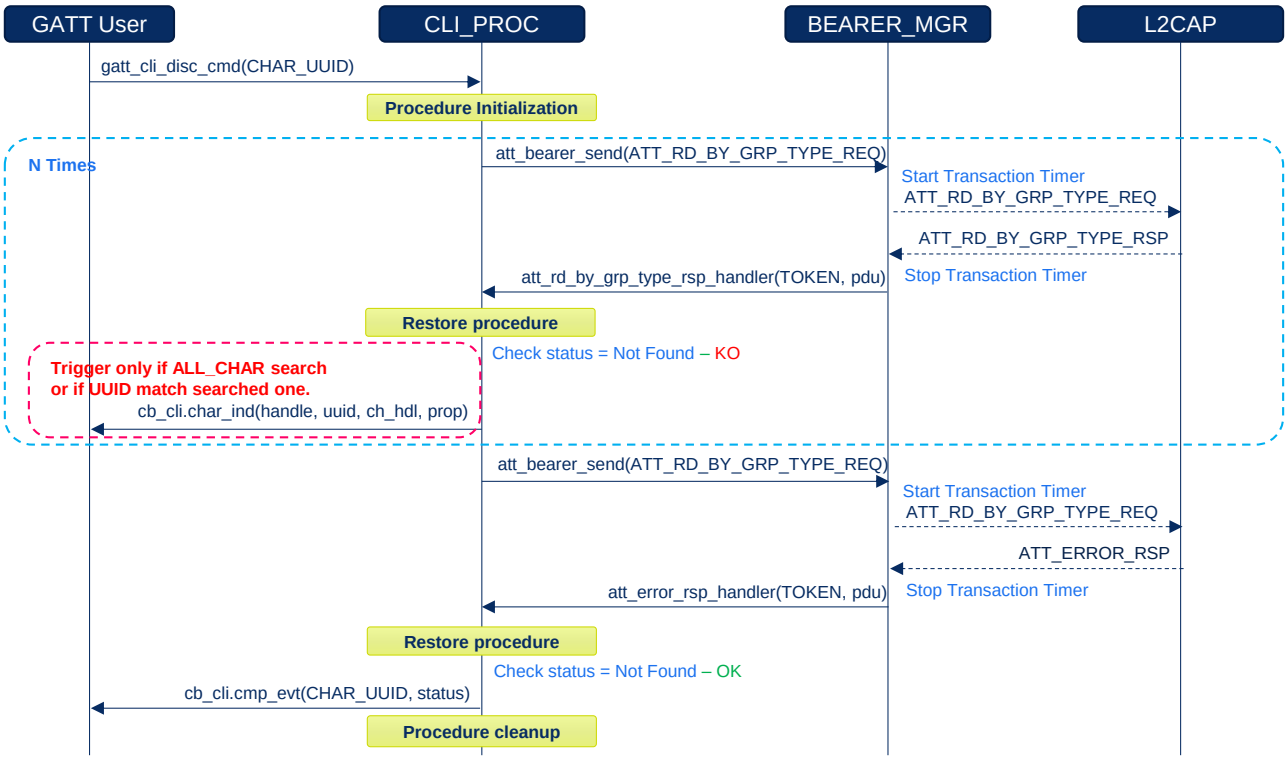

Figure 6-15: Discover peer characteristics (all or with specific UUID) MSC

Note: Same procedure is used to discover all characteristics or with a specific UUID. The filtering of UUID is performed by client side and not by service side.

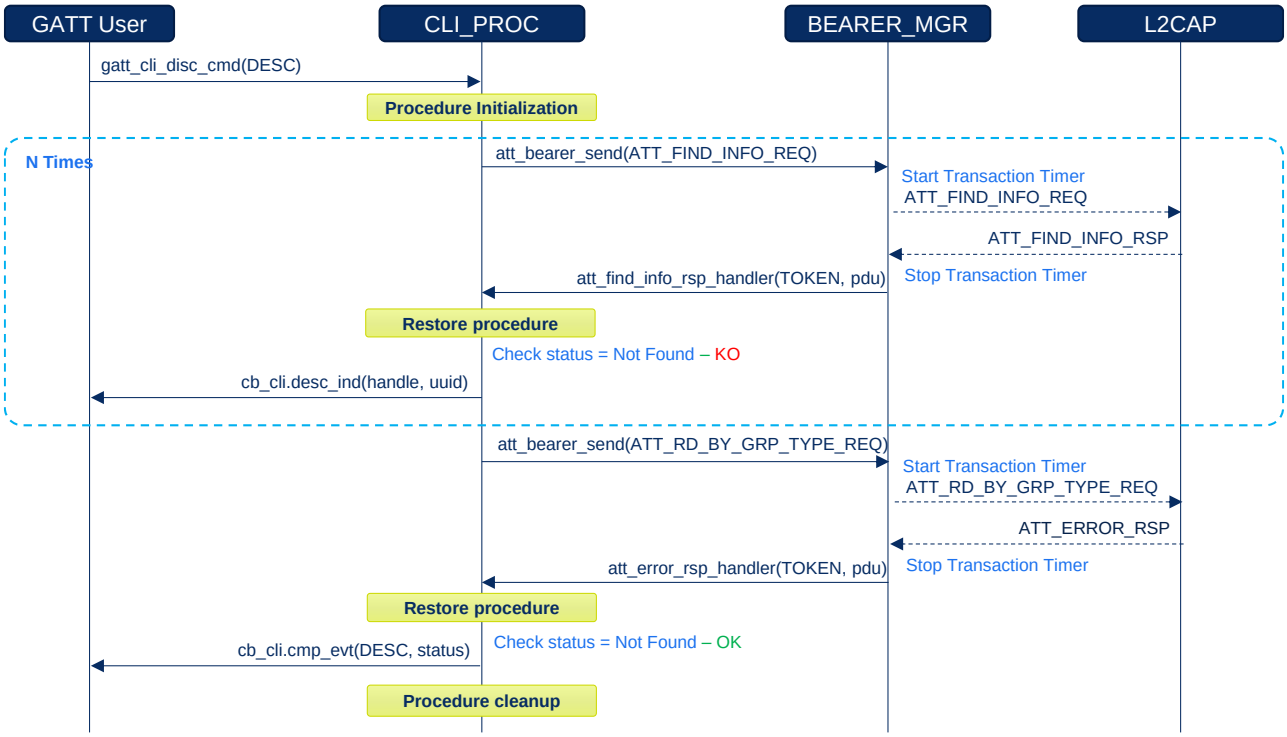

Figure 6-16: Discover peer descriptors MSC

6.2.3 Read Command

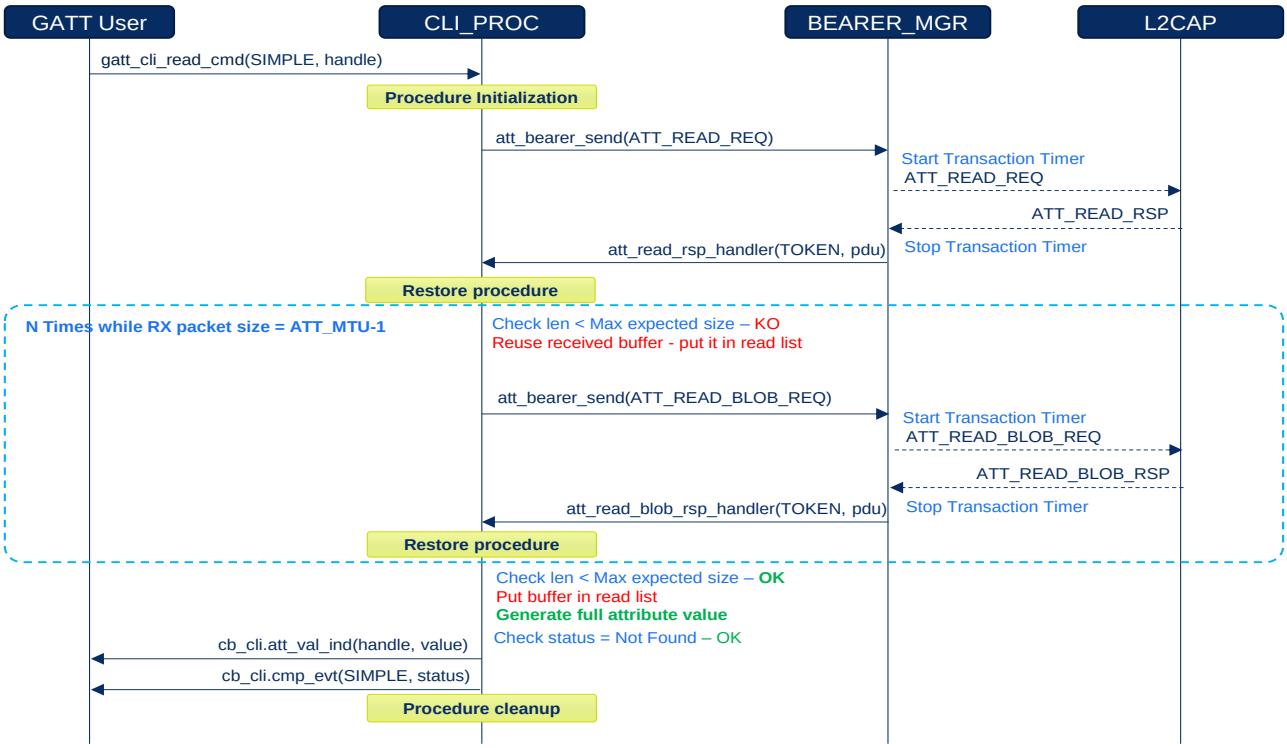

Figure 6-17: Read Simple Request MSC

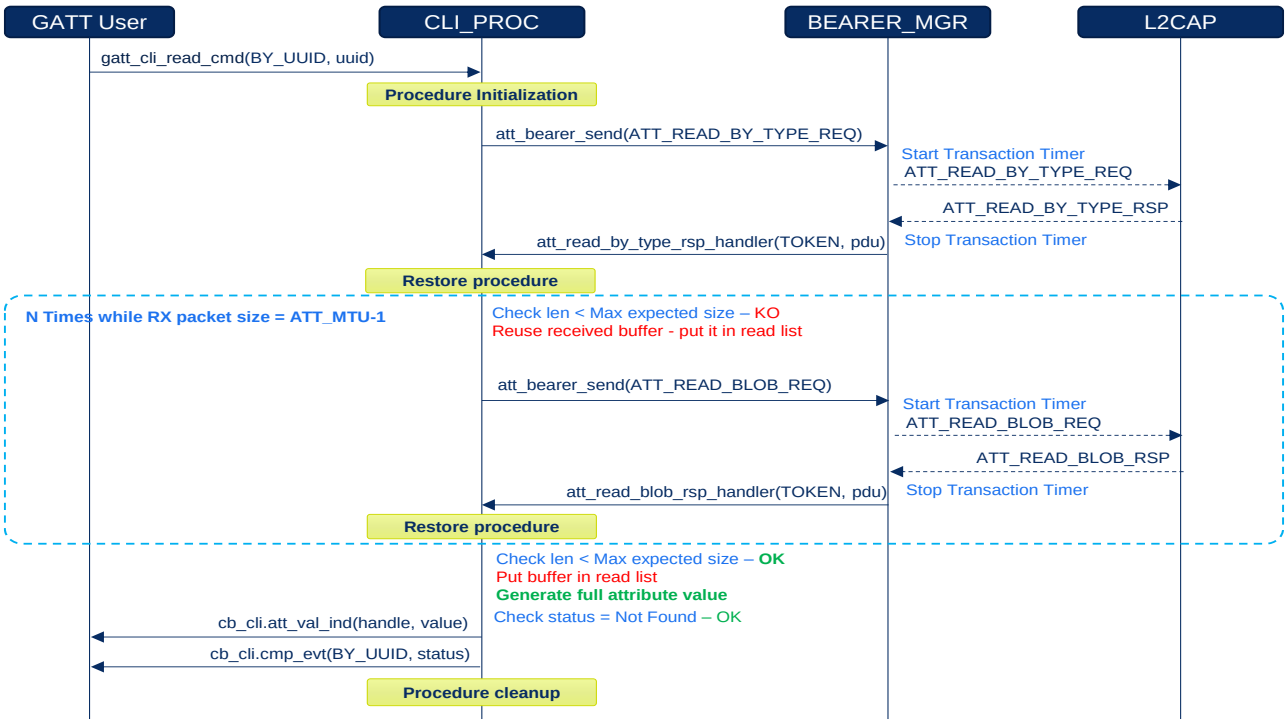

Figure 6-18: Read By UUID Request MSC

6.2.4 Write Command

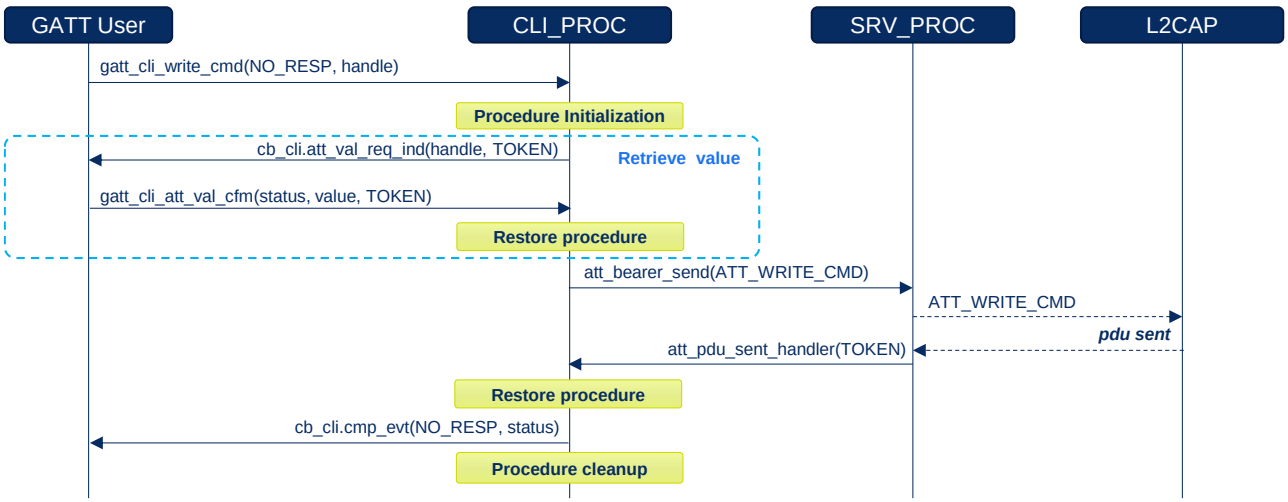

Figure 6-19: Write Command MSC

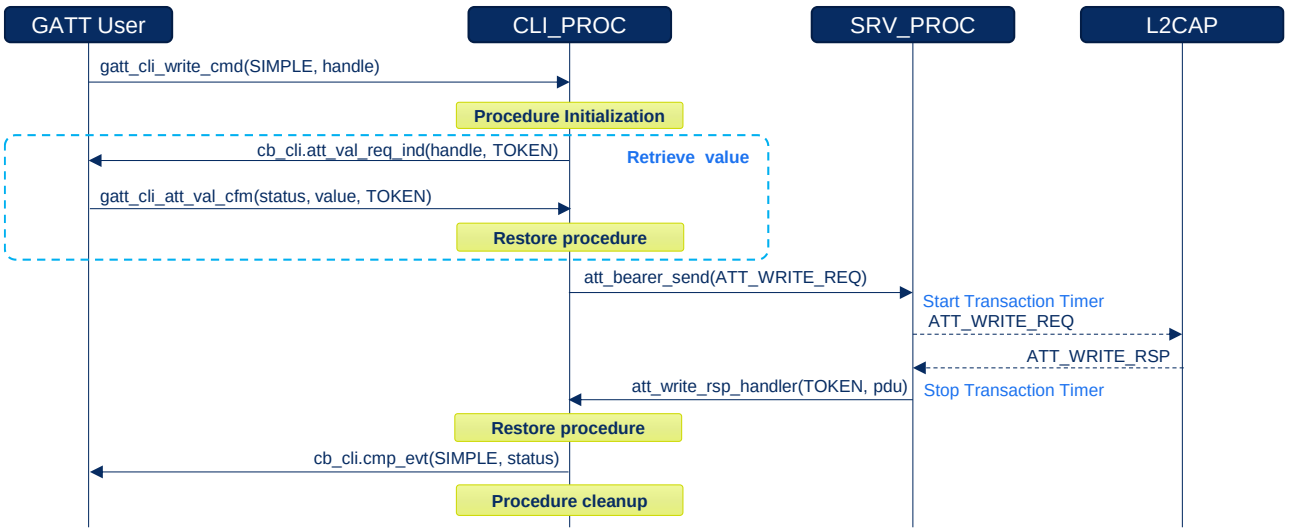

Figure 6-20: Write Request MSC

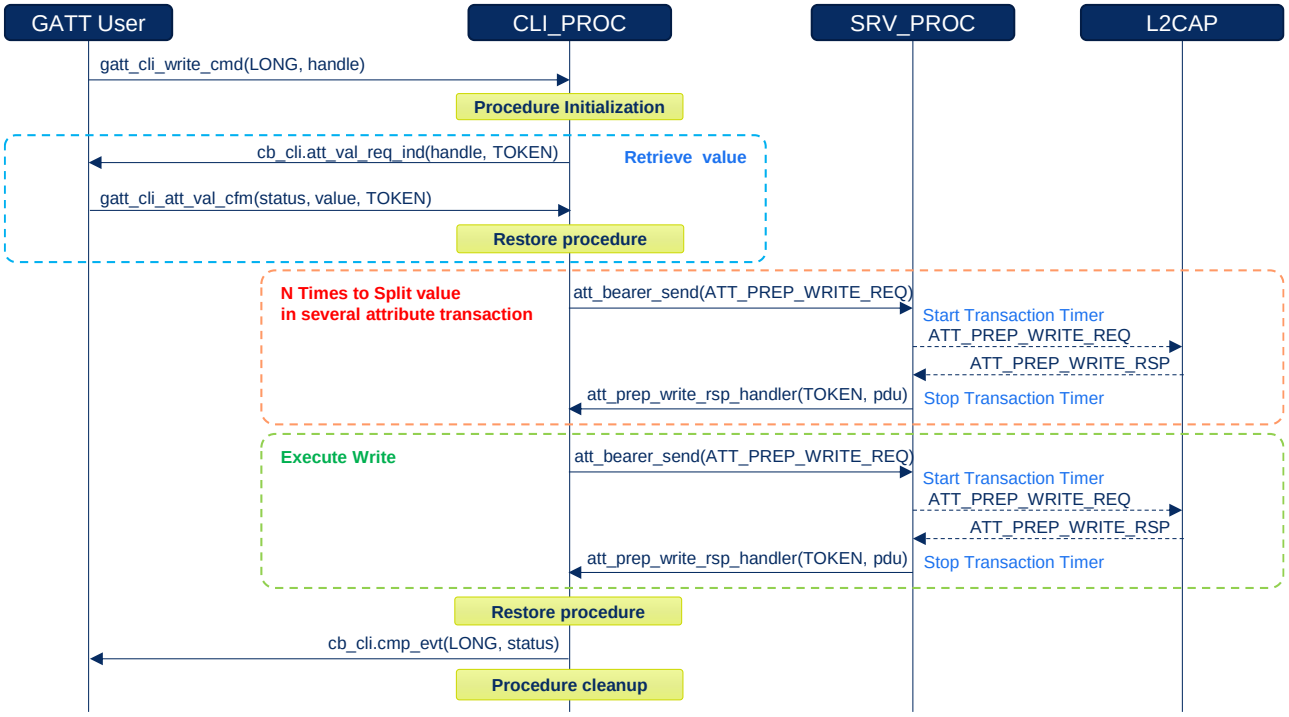

Figure 6-21: Write Long/Multiple MSC

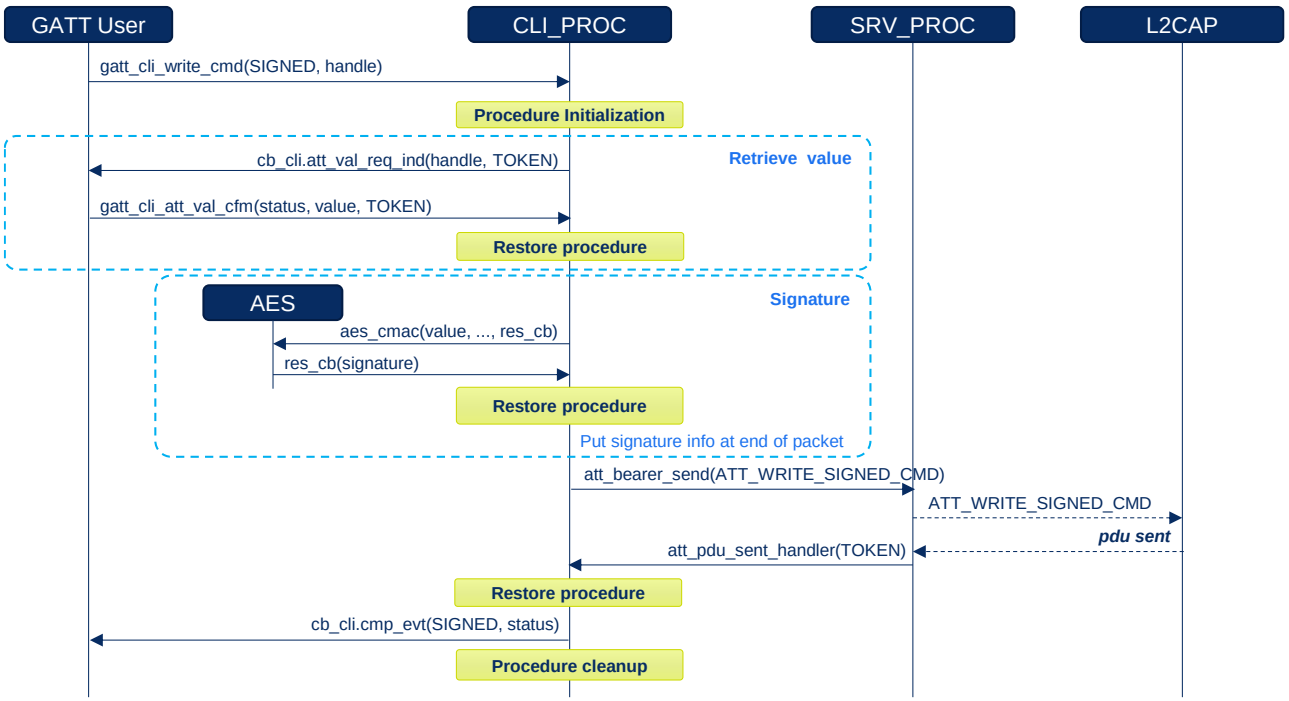

Figure 6-22: Write Signed MSC

6.2.5 Service Discovery Procedure

The Service Discovery Procedure is a generic feature that can be used by any upper layer in order to discover content of peer device's database.
By using a generic method of service discovery it prevent from code duplication in client profiles.
This discovery should be performed for all services type or only for some of them.
With this feature, application can decide if discovery is performed by client profiles or by application itself.
For each discovered services, this procedure is in charge of finding included services, characteristics and descriptors.
When a full service is discovered, this operation triggers a message containing all information.

Note: Since this procedure can be very long, it could be aborted by application through a Cancel API.

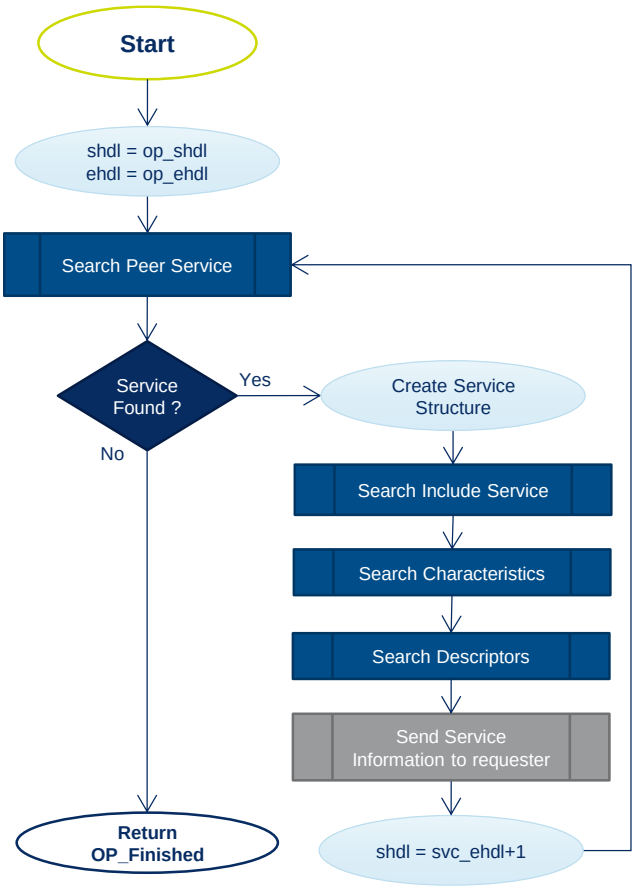

Figure 6-23: Service Discovery procedure state machine.

Nb hdl				
svc_handle	UUID			
att_handle	INC_SVC	UUID		
att_handle	CHAR	prop	handle	UUID
att_handle	UUID			
att_handle	CCC			
att_handle	CHAR	prop	handle	UUID
att_handle	UUID			
att_handle	descriptor			

Figure 6-24: Overview of information present in discovered service.

7 Robust caching

The robust caching feature requires some information to be kept, so that it can be reused upon reconnection with bonded devices.

On server side, when a connection is established, application is informed about update of client knowledge of local data base status:

- When local database is updated, peer client is considered unaware of this change
- Due to attribute exchange, client becomes change aware

If content of local database has been updated, application must also keep in mind that bonded peer devices with which there is no active connection will have to be notified about this update upon reconnection.

On client side, content of peer device database has to be stored as well as attribute handle of the Service Changed characteristic value attribute. There is an automatic procedure to enable robust caching, register to service changed indication and read peer database hash.

List of Acronyms and Abbreviations

Acronym or abbreviation	Writing out in full / Meaning
ATT	Attribute Protocol
BLE	Bluetooth Low Energy. Also termed LE.
CSRK	Connection Signature Resolving Key
FS	Functional Specification
FW	Firmware
GAP	Generic Access Profile
GATT	Generic Attribute Profile
HCI	Host Controller Interface
L2CAP	Logical Link Control and Adaptation Protocol
LE	(Bluetooth) Low Energy
LE_PSM	Low Energy Protocol/Service Multiplexer
LE_COC	Low Energy Connection Oriented Channel
LL	Lower Layer (Link Manager and Link Controller)
MTU	Maximum Transmission Unit
PDU	Protocol Data Unit
PSM	Protocol/Service Multiplexer
RO	Read Only
SC	Secure Connection
SDU	Service Data Unit
SIG	Special Interest Group
SMP	Security Manager Protocol. BLE block responsible for security.
UUID	Universally Unique Identifier

References

[1]	**Title**	Specification of the Bluetooth System v5.1
	Reference	Bluetooth Specification LE LL and BR/EDR
	Source	Bluetooth SIG

[2]	**Title**	RivieraWaves Kernel
	Reference	RW-BT-KERNEL-SW-FS
	Source	RivieraWaves

[3]	**Title**	RW HCI Software
	Reference	RW-HCI-SW-FS
	Source	RivieraWaves

[4]	**Title**	GAP Interface Specification
	Reference	RW-BLE-GAP-IS
	Source	RivieraWaves

[5]	**Title**	GATT Interface Specification
	Reference	RW-BLE-GATT-IS
	Source	RivieraWaves

[6]	**Title**	RW-BLE Link Layer Software
	Reference	RW-BLE-LL-SW-FS
	Source	RivieraWaves