


Bluetooth WHITE PAPER	DATE 01 July 99	N.B.	DOCUMENT NO 1.C.118/1.0
RESPONSIBLE Brent Miller	E-MAIL ADDRESS bamiller@us.ibm.com		STATUS

Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer

Version 1.0



Bluetooth provides a Service Discovery layer and defines a series of primitives to access the functions of the Service Discovery layer. This paper describes a mapping between these primitives and the APIs and functions in the Salutation Architecture.

Special Interest Group (SIG)

The following companies are represented in the Bluetooth Special Interest Group:
Ericsson Mobile Communications AB

IBM Corp.

Intel Corp.

Nokia Mobile Phones

Toshiba Corp.

Disclaimer and copyright notice

THIS DRAFT DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document.

Copyright © IBM Corp., 1999. *Third-party brands and names are the property of their respective owners.

Revision History

Revision	Date	Comments
0.1	3/29/1999	First Draft
0.9	6/1/99	Response to SIG Comments
1.0	7/01/99	Final Version 1.0

Contributors

Robert Pascoe

Salutation Consortium

Brent Miller

IBM

Contents

1	Overview	5
2	Bluetooth Summary	6
2.1	Bluetooth Service Discovery Protocol	6
2.1.1	Protocol Data Unit Types.....	7
2.2	Bluetooth Profile Stack.....	8
2.3	Bluetooth Service Primitives.....	8
3	Salutation Architecture Summary.....	10
3.1	Service Broker Tasks	11
3.1.1	Service Registry	11
3.1.2	Service Discovery.....	11
3.2	Salutation Manager API Specification	12
3.2.1	Salutation Manager API Description.....	12
3.3	Service Discovery Flow	14
4	Mapping Bluetooth SDP to Salutation APIs.....	15
4.1	General Mapping Assumptions	15
	Salutation API Mapping.....	16
4.2.1	Configuration	16
4.2.2	Mapping.....	16
4.2.3	Summary	19
4.3	Salutation Manager Mapping	19
4.3.1	Configuration	19
4.3.2	Mapping.....	20
4.3.2.1	Capability Search.....	20
4.3.2.2	Capability Query.....	22
4.3.3	Stop Rules.....	23
4.3.4	Summary	23
5	References.....	25
6	Definitions.....	26

1 Overview

The Bluetooth protocol stack contains a service discovery protocol (SDP) [1] that enables the retrieval of information that can be used to configure the stack to support several end-user applications. SDP can further be used to locate services that are available on devices in the vicinity of the user. Having located available services, a user may then select to use any of them.

SDP provides direct support for the following set of service inquiries:

- search for services by service class;
- search for services by service attributes; and
- service browsing.

A service discovery profile is provided [2] that describes a generic syntax and semantics to be used by a service discovery application to locate services in other processes using Bluetooth SDP. The primitives are described in a generic way as these primitives may be operating environment dependent.

The Salutation Architecture [3] provides a standard method for applications, services and devices to describe and to advertise their capabilities to other applications, services and devices and to find out their capabilities. The architecture also enables applications, services and devices to search other applications, services or devices for a particular capability, and to request and establish interoperable sessions with them to utilise their capabilities.

This paper maps Bluetooth service discovery to the Salutation Architecture. Specifically this paper (1) maps the Bluetooth service discovery profile to the Salutation APIs and (2) maps the Bluetooth Service Discovery Protocol to the Salutation Manager.

2 Bluetooth Summary

2.1 Bluetooth Service Discovery Protocol

Figure 2.1 shows the Bluetooth protocols and supporting entities.

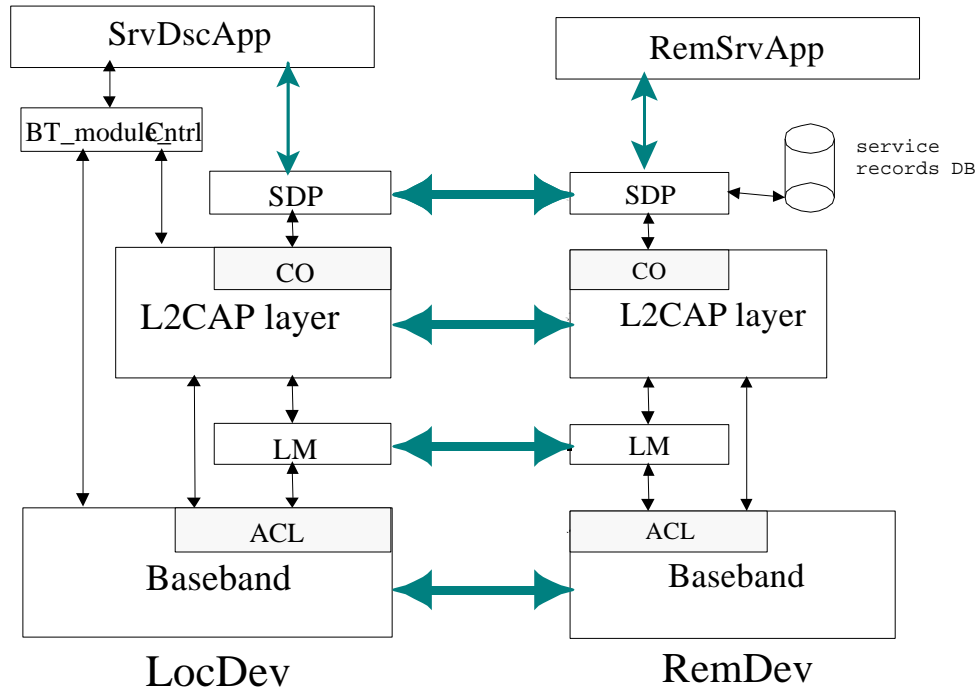


Figure 2.1: The Bluetooth protocol for the service discovery profile

The blocks marked SDP indicate the service discovery component. An SDP block generates and receives Bluetooth **service discovery protocol** (SDP) commands and responses from the lower layers of the Bluetooth stack.

SDP provides a means for client applications to discover the existence of services provided by server applications as well as the attributes of those services. The attributes of a service include the type or class of service offered and the mechanism or protocol information needed to utilise the service.

As shown in Figure 2.1, SDP involves communication between an SDP server located on RemDev and an SDP client located on LocDev. The server maintains a list of service records that describe the characteristics of services associated with the server. Each service record contains information about a single service.

A client may retrieve information from a service record maintained by the SDP server by issuing an SDP request.

All of the information about a service is maintained by an SDP within a single **service record**. The service record consists entirely of a list of **service attributes**. Each service attribute describes a single characteristic of a service.

SDP uses a request/response model where each transaction consists of one request **protocol data unit** (PDU) and one response PDU. Generally, each type of request PDU has a corresponding type of response PDU. However, if the server determines that a request is improperly formatted or for any reason the server cannot respond with the appropriate PDU type, it will respond with an SDP_ErrorResponse PDU.

2.1.1 Protocol Data Unit Types

- **ServiceSearch Transaction:**

The SDP client generates a SDP_ServiceSearchRequest to locate service records that match the service search pattern given as the first parameter of the PDU. Upon receipt of this request, the SDP server will examine its service record data base and return an SDP_ServiceSearchResponse containing the service record handles of service records that match the given service search pattern.

- **ServiceAttribute Transaction:**

The SDP client generates a SDP_ServiceAttributeRequest to retrieve specified attribute values from a specific service record. The service record handle of the desired service record and a list of desired attribute ids to be retrieved from that service record is supplied as parameters.

- **ServiceSearchAttribute Transaction:**

The SDP_ServiceSearchAttributeRequest transaction combines the capabilities of the SDP_ServiceSearchRequest and the SDP_ServiceAttributeRequest into a single request. As parameters, it contains both a service search pattern and a list of attributes to be retrieved from service records that match the service search pattern. The SDP_ServiceSearchAttributeRequest and its response are more complex and may require more bytes than separate SDP_ServiceSearch and SDP_ServiceAttribute transactions. However, using SDP_ServiceSearchAttributeRequest may reduce the total number of SDP transactions, particularly when retrieving multiple service records.

- **Browsing for Services:**

Normally, a client searches for services based on some desired characteristic(s) of the services. However, there are times when it is desirable to discover which types of services are described by an SDP server's service

records without any *a priori* information about the services. This process of looking for *any* offered services is termed browsing. In SDP, the mechanism for browsing for services is based on an attribute shared by all service classes. This attribute is called the *BrowseGroupList* attribute. Each attribute represents a *browse group* with which a service may be associated for the purpose of browsing. When a client desires to browse an SDP server's services, it creates a service search pattern containing the attribute that represents the *root browse group*.

Refer to Reference [1] for details about the Bluetooth service attribute definitions.

2.2 Bluetooth Profile Stack

Referring to Figure 2.1, the service discovery user application (SrvDscApp) in a local device (LocDev) interfaces with the SDP protocol to send service inquires and receive service inquire responses from other remote devices (RemDev). The SDP uses the connection-oriented (CO) transport service in L2CAP, which in turn uses the baseband asynchronous connectionless (ACL) links to carry ultimately the SDP PDUs over the air.

2.3 Bluetooth Service Primitives

This section briefly describes the service primitives that the Bluetooth stack needs to expose to the SrvDscApp to perform its task.

Table 2.1 contains a minimum set of enabling service primitives to support a SrvDscApp. Different implementations of the Bluetooth stack shall (at a minimum) enable the functions that these service primitives provide. For example, the **serviceSearch()** service primitive permits multiple identical operations to be handled at once. A stack implementation that requires an application to accomplish this function by iterating through the multiple identical operations one at a time will be considering as enabling the function of this service primitive.

service primitive	function accomplished
serviceBrowse (LIST(<i>RemDev</i>); LIST(<i>RemDevRelation</i>); LIST(<i>browseGroup</i>); <i>stopRule</i>)	searches for services (service browsing) that belong to the list of <i>browseGroup</i> services in the devices in the list of <i>RemDevs</i> ; the search may be further qualified with a list of <i>RemDevRelation</i> parameters, whereby a user specifies the trust and connection relation of the devices to be searched, e.g., search only the devices that are in the <i>RemDev</i> list for which pairing has been performed; search continues until the stopping rule <i>stopRule</i> is satisfied
serviceSearch (LIST(<i>RemDev</i>); LIST(<i>RemDevRelation</i>); LIST(<i>searchPath</i> ,	searches whether the devices listed in the list of <i>RemDevs</i> support services in the requested list of services; each service in the list must have a service search path that is a superset of the <i>searchPath</i> ; for each such service the values of the attributes contained

<pre> attributeList); stopRule) </pre>	<p>in the corresponding <i>attributeList</i> are also retrieved; the search may be further qualified with a list of <i>RemDevRelation</i> parameters, whereby a user specifies the trust and connection relation of the devices to be searched, e.g., search only the devices that are in the <i>RemDev</i> list for which pairing has been performed; search continues until the stopping rule <i>stopRule</i> is satisfied</p>
<pre> enumerateRemDev (LIST(<i>classOfDevice</i>); stopRule) </pre>	<p>searches for RemDev in the vicinity of a LocDev; RemDev searches may optionally be filtered using the list of <i>classOfDevice</i>, e.g., LAN APs; search continues until the stopping rule <i>stopRule</i> is satisfied</p>
<pre> getRemDevName (LIST(<i>primitiveHandle</i>); stopRule) </pre>	<p>retrieves the names of devices associated with the execution of the service primitives identified by the list of <i>primitiveHandle</i>;¹ search continues until the stopping rule <i>stopRule</i> is satisfied</p>
<pre> terminatePrimitive (<i>primitiveHandle</i>; returnResults) </pre>	<p>terminates the actions executed as a result of invoking the services primitive identified by the <i>primitiveHandle</i>; optionally, this service primitive may return any partially accumulated results related to the terminated service primitive</p>

Table 2.1: Service primitives in support of SrvDscApp

¹ It is assumed that each invocation of a service primitive can be identified by a *primitiveHandle* the realization of which is implementation dependent.

3 Salutation Architecture Summary

The **Salutation Architecture** was created to solve the problems of **service discovery and utilization** among a broad set of appliances and equipment and in an environment of widespread connectivity and mobility.

The architecture provides a standard method for applications, services and devices to describe and to advertise their capabilities to other applications, services and devices and to find out their capabilities. The architecture also enables applications, services and devices to search other applications, services or devices for a particular capability, and to request and establish interoperable sessions with them to utilize their capabilities.

Given the diverse nature of target appliances and equipment in an environment of widespread connectivity, the architecture is processor, operating system, and communication protocol independent, and allows for scalable implementations, even in very low-price devices.

As shown in Figure 3-1, the Salutation Architecture defines an entity called the **Salutation Manager (SLM)** that functions as a service broker for applications, services and devices called a Networked Entity. The Salutation Manager allows Networked Entities to discover and utilize the capabilities of other Networked Entities.

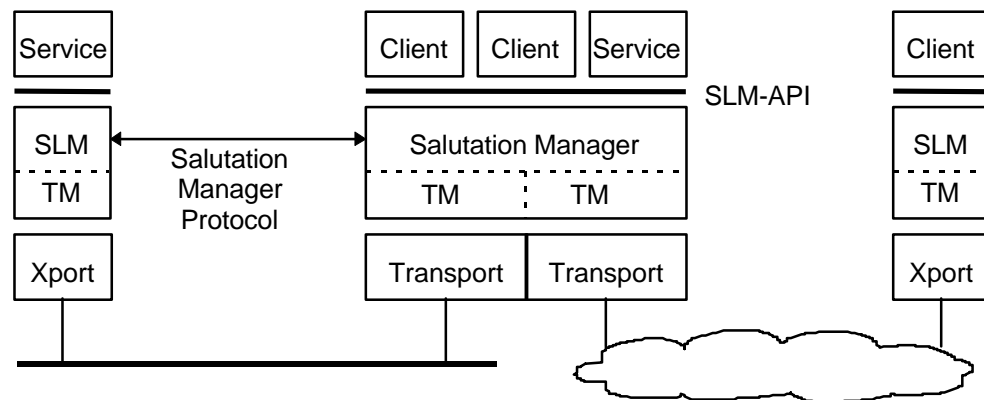


Figure 3-1: Model of the Salutation Manager

A Networked Entity may be a service provider, called a **Service**. The Service registers its capability with a Salutation Manager. A Networked Entity may be a service user, called a **Client**. The Client discovers Services and requests to use them through a Salutation Manager. A Networked Entity may serve as either a Client or a Service, or both.

The Salutation Manager provides a transport-independent interface, called the **Salutation Manager Application Program Interface (SLM-API)**, to Services and Clients. The architecture defines an abstract procedural SLM-API.

The Salutation Manager communicates with other Salutation Managers to perform its role as a service broker. The Salutation Manager-to-Salutation Manager communications protocol is defined by the Salutation Architecture and called the **Salutation Manager Protocol**. The Salutation Manager binds to a specific transport through a **Transport Manager (TM)** unique to that transport class.

3.1 Service Broker Tasks

To perform its function as service broker, the Salutation Manager provides four basic tasks:

- Service Registry
- Service Discovery
- Service Availability
- Service Session Management

Service Registry and Service Discovery, and the APIs which expose these functions to the Client/Server layer, are of primary interest for this document.

3.1.1 Service Registry

The Salutation Manager contains a **Registry**² to hold information about Services². The minimum requirement for the Registry is to store information about Services connected to the Salutation Manager. Optionally, the Salutation Manager Registry may store information about Services that are registered in other Salutation Managers. All requests by other equipment for Salutation resources would be directed toward other Salutation Managers which would respond accordingly.

The limit on Registry implementation is the size of the storage reserved for the Registry function.

3.1.2 Service Discovery

The Salutation Manager can discover other remote Salutation Managers and determine the Services registered there. **Service Discovery** is performed by comparing a required Services type(s), as specified by the local Salutation Manager, with the Service type(s) available on a remote Salutation Manager. Through manipulation of the specification of required Service type(s), the Salutation Manager can determine:

- The characteristics of **all the Services** registered at a remote Salutation Manager

² Equivalent to Bluetooth service record DB shown in Figure 2.1

- The characteristics of a **specific Service** registered at a remote Salutation Manager
- The presence of a Service on a remote Salutation Manager **matching a specific set of characteristics**.

3.2 Salutation Manager API Specification

This section describes an abstract definition of the SLM-API, the application programming interface provided by the Salutation Manager to Salutation applications. More specifically, this section focuses on the Salutation Service Registration and Service Discovery APIs. It is intended to provide a level of understanding to aid in the mapping of Salutation APIs to Bluetooth SDP functions. The SLM-API supporting the mapping are:

- **Service Registration**

slmRegisterCapability()
slmUnregisterCapability()

- **Service Discovery**

slmSearchCapability()
slmQueryCapability()

3.2.1 Salutation Manager API Description

The SLM-APIs are described in the abstract in this section.

When the Client calls the Salutation Manager through the SLM-API, it is called the **local Salutation Manager**. Any other Salutation Manager is called a **remote Salutation Manager**. Refer to Reference [3] for details of the API attributes of SLM-ID, Functional Unit Description Record, and Service Description Record.

Abstract SLM-API	function accomplished
<p><i>slmRegisterCapability()</i></p> <p>Input Parameters</p> <p>(</p> <p> Functional Unit Description Record³;</p> <p> Callback Entry for Open Service Indication;</p> <p> Callback Entry for Close Service</p>	<p>The <i>slmRegisterCapability()</i> function is called by Services to register their specific instances of Functional Units with the local Salutation Manager. The specific instance is described in a record called a Functional Unit Description Record. The calling Service passes a Functional Unit Description Record, which describes its capability, to the Salutation Manager. The Salutation Manager returns a Functional Unit Handle that uniquely identifies the Functional Unit among all the Functional Units registered with the Salutation Manager.</p>

³ The Salutation Architecture defines the Functional Unit Description Record as a record that identifies the Functional Unit, and the capabilities of that instance of the Functional Unit. The Functional Unit maps to the Bluetooth Service and capabilities map to the Bluetooth Attributes.

<p>Indication; Callback Entry for Receive Data Indication; Preferred Functional Unit Handle)) Output Parameter (Functional Unit Handle)</p>	<p>While a Service has a Functional Unit registered with a local Salutation Manager, the Functional Unit's capability may be included in the response to a <i>Query Capability</i> request.</p> <p>The Callback Entries are provided to provide an entry point into a Service when that service is to be used. Entry points are provided for opening and closing the Service as well as for receiving data.</p> <p>The Service may attempt to specify a handle for the functional unit instance being registered. This value will be assigned if it is not currently in use. Otherwise, the Salutation manager will assign a random, unused value for the handle.</p>
<p><i>slmUnregisterCapability()</i> Input Parameters (Functional Unit Handle) Output Parameter None</p>	<p>The Service, which has registered itself with the local Salutation Manager by calling the <i>slmRegisterCapability()</i> function, calls this function to unregister itself from the local Salutation Manager.</p> <p>The Functional Unit Handle is the value returned by the <i>slmRegisterCapability()</i> used to register this Service.</p>
<p><i>slmSearchCapability()</i> Input Parameters (SLM-ID; Service Description Record⁴; Output Parameter (List of SLM-IDs)</p>	<p>The Client calls this function to ask the local Salutation Manager to search for Salutation Managers having a registered Functional Unit with a specific capability. The local Salutation Manager returns the list of SLM-IDs to the Client. Salutation Manager(s) whose SLM-ID(s) are included in the list has(have) a Functional Unit(s) that can provide the Service requested by the Client.</p> <p>SLM-ID is NULL for version 2.0 of the Salutation Architecture</p> <p>Service Description Record describes the Service(s) and their capabilities that are of interest to the Client. A Service Description Record that contains a Functional Unit Description Record of "All Call" Functional Unit ID with no Attribute Records, may be specified to get the list of all the SLM-IDs of Salutation Managers known to the local Salutation Manager.</p>
<p><i>slmQueryCapability()</i> Input Parameters (SLM-ID; Service Description Record)</p>	<p>The Client calls this function to discover registered Functional Units and their capabilities at a specific Salutation Manager.</p> <p>SLM-ID specifies the target Salutation Manager. If NULL is specified, the target Salutation Manager is the local Salutation Manager.</p> <p>The Input Service Description Record describes the Service(s) and their capabilities that are of interest to</p>

⁴ The Salutation Architecture defines the Service Description Record as a collection of one or more Functional Unit Description Records. The Service Description Record describes all the services sought by a Client or all the services maintained by a Service.

Output Parameter	the Client.
Service Description Record)	The Output Service Description Record describes the Service(s) and their capabilities that match the Input Service Description Record.

3.3 Service Discovery Flow

The flow of Remote Service Discovery messages and calls are depicted in Table 3-4. Salutation APIs are used by the Client and Functional Unit to access their respective Salutation Managers. Salutation Protocol flows between the Client-side and Server-side Salutation Managers.

Client	Client-side Salutation Manager	Salutation Protocol	Service-side Salutation Manager	Functional Unit
			<== slmRegisterCapability() call slmRegisterCapability() return ==>	
	slmSearchCapability() call ==>			
	Query Capability call ==> <== Query Capability reply : (This step is repeated for each known SLM. The reply data maybe cached for the next step.)			
	<== slmSearchCapability() return			
	slmQueryCapability() call ==>			
	Query Capability call ==> <== Query Capability reply : (This step is optional, depending on the caching capability of the Client's Salutation Manager.)			
	<== slmQueryCapability() return : (This step is repeated for each Salutation Manager found by the Search Capability. The Salutation Manager returns the cached data.)			
			<== slmUnRegisterCapability() call slmUnRegisterCapability() return ==>	

Table 3-4: Remote Service Discovery Flow Diagram

4 Mapping Bluetooth SDP to Salutation APIs

Two approaches will be used for mapping Bluetooth SDP primitives to Salutation APIs.

The first approach will assume that the Salutation APIs are implemented on top of the Bluetooth service discovery. In this case, the mapping will show how SDP attributes can be passed in the Salutation APIs. That is:

Salutation APIs → Bluetooth SDP → Bluetooth Protocol

Here, the Salutation APIs are implemented as the entry to Bluetooth SDP. SDP extracts the information it requires from the APIs and processes according to the mapping to SDP primitives.

The second approach will assume that Salutation Manager can be map directly to the SDP protocol using a Bluetooth specific Transport Manager (indicated by TM in Figure 3.1). That is:

Salutation APIs → Salutation Manager → Bluetooth Protocol

Here, SDP is replaced by the Salutation Manager, with the Salutation Manager mapping its functionality to SDP protocol.

4.1 General Mapping Assumptions

- SDP is a service manager. For RemDev, it provides the ability to specify local services and respond to requests to discover the services it manages. For LocDev, it provides the ability for SvcDscApp to ask RemDev if it supports specific services. The APIs provide a means for application developers to access the functions of the SDP service manager.
- Service requests by LocDev are accessed through Salutation ***slmSearchCapability()*** and ***slmQueryCapability()*** API calls.
- Certain Bluetooth RemDevs will have the need to dynamically update the services they support. That is, a RemDev may need to update the service records maintained in the service record DB. Salutation ***slmRegisterCapability()*** and ***slmUnregisterCapability()*** API calls will

be mapped on the RemDev side to support dynamic registry of services.

4.2 Salutation API Mapping

This section describes how Salutation APIs can be used to represent the SDP primitives.

4.2.1 Configuration

The configuration used for this mapping is shown in Figure 4.1. The figure shows the use of the Salutation APIs as the entry point to SDP. No other changes have been made to the Bluetooth model shown in Figure 2.1.

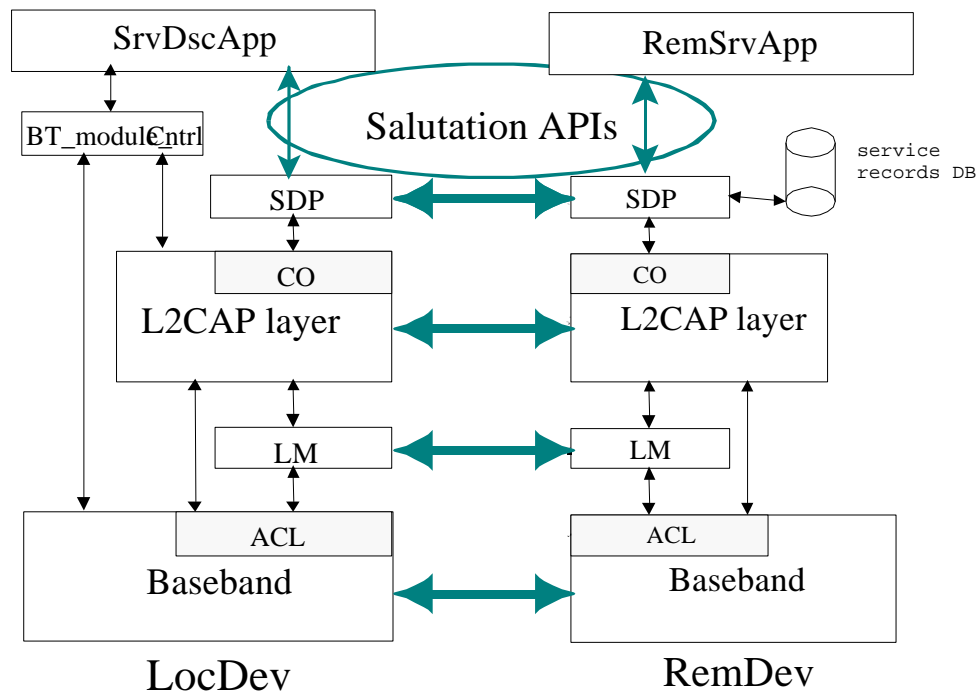


Figure 4.1: Salutation API Mapping to Bluetooth SDP

4.2.2 Mapping

Table 4.1 depicts the general mapping of SDP primitives to Salutation APIs. Although there are no parallels defined in the SDP primitives, the Salutation Register Capability and Unregister Capability APIs are included for completeness.

The function provided by the SDP `getRemDevName` - the return of the names of devices identified by the SDP service searches - is an integral part of the Salutation's `slmSearchCapability()` and `slmQueryCapability()` calls. As a result, one Salutation API can provide the function of two SDP primitives.

This mapping uses the functional definition of the Salutation APIs, but not the parameter values. The APIs becomes a vehicle for passing the SDP parameters to the Bluetooth SDP Manager. That is, the SLM-ID and Service Description Records parameters of the Salutation APIs are replaced with the appropriate SDP parameter values. Therefore, the parameters defined by the SDP primitives are passed without modification to the Bluetooth SDP Manager via the Salutation API format. The parameter returned from the search operations is a list of the names of the devices identified by the search.

SDP Service Primitive	Salutation Primitive
ServiceBrowse, getRemDevName	SlmQueryCapability() Input Parameters (LIST(<i>RemDev</i>); LIST(<i>RemDevRelation</i>); LIST(<i>browseGroup</i>); <i>stopRule</i>) Output Parameter (List of Device Names)
ServiceSearch, getRemDevName	SlmQueryCapability() Input Parameters (LIST(<i>RemDev</i>); LIST(<i>RemDevRelation</i>); LIST(<i>searchPath</i> , <i>attributeList</i>); <i>stopRule</i>) Output Parameter (List of Device Names)
EnumerateRemDev, getRemDevName	SlmSearchCapability() Input Parameters (LIST(<i>classOfDevice</i>); <i>stopRule</i>)

	<p>)</p> <p>Output Parameter</p> <p>(</p> <p>List of Device Names</p> <p>)</p>
(No SDP Registration Primitives)	<p>slmRegisterCapability()</p> <p>Input Parameters</p> <p>(</p> <p>LIST(<i>attributeList</i>);</p> <p>Callback Entry for Open Service Indication;</p> <p>Callback Entry for Close Service Indication;</p> <p>Callback Entry for Receive Data Indication;</p> <p>Preferred Functional Unit Handle</p> <p>)</p> <p>Output Parameter</p> <p>(</p> <p>Functional Unit Handle</p> <p>)</p>
(No SDP Registration Primitives)	<p>slmUnregisterCapability()</p> <p>Input Parameters</p> <p>(</p> <p>Functional Unit Handle</p> <p>)</p> <p>Output Parameter</p> <p>None</p>

Table 4.1 SDP primitive to Salutation API mapping

The `slmSearchCapability()` call is used for both the `serviceBrowse` and the `serviceSearch` SDP primitives. The differentiator is the presence or absence of the `browseGroup` list parameter. If this parameter is present, the Bluetooth SDP Manager performs a browse operation. Otherwise a search operation is performed.

The Functional Unit Description Record parameter of the Salutation `slmRegisterCapabilities` API is replaced with the SDP `attributeList` parameter that specifies the capabilities of the service being registered. The callback parameters remain in the API definition, providing a means to define the entry points for service utilization. The returned value remains a handle of the registered service.

This value is used in the `slmUnregisterCapability` API to identify the service to be removed from public access.

4.2.3 Summary

The Salutation API mapping provides a means to pass SDP primitive attributes to the Bluetooth SDP Manager.

4.3 Salutation Manager Mapping

This section describes how the Salutation Manager, accessed via the Salutation APIs, can be used to generate Bluetooth SDP protocol.

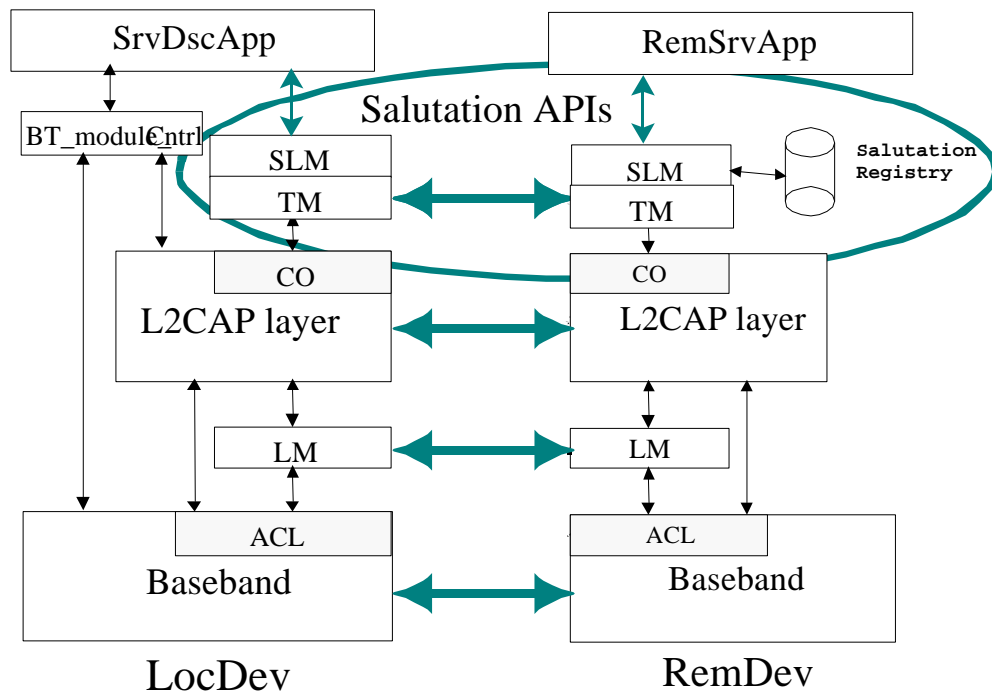


Figure 4.2: Salutation Manager Mapping to Bluetooth SDP protocol

4.3.1 Configuration

The configuration used for this mapping is shown in Figure 4.2. The figure shows the use of the Salutation Manager (SLM) in both LocDev and RemDev to provide the service the management functionality of SDP. SLM exposes the existing Salutation APIs to the `SrvDscApp` and the `RemSrvApp`. SLM generates the appropriate SDP protocol through its TM, handing it off to L2CA layer. SLM also responds to SDP protocol received from L2CA. No other changes have been made to the Bluetooth model shown in Figure 2-1.

4.3.2 Mapping

Table 4.2 depicts the general mapping of Salutation Manager functions to the SDP protocol. LocDev and RemDev use the Salutation APIs to access their respective Salutation Managers. Bluetooth Service Discovery Protocol flows between the LocDev and RemDev Salutation Managers.

Two transformations exist between the Salutation API layer and the Bluetooth SDP protocol layer. The Salutation Manager provides these transformations.

`slmSearchCapability()` \leftrightarrow `SDP_ServiceSearch`

`slmQueryCapability()` \leftrightarrow `SDP_ServiceSearch` & `SDP_ServiceAttribute`

4.3.2.1 Capability Search

The Salutation `slmSearchCapability()` API call is mapped by the Salutation Manager's TM to a Bluetooth `ServiceSearch` protocol. The Salutation Manager maps the Functional Unit Description Record passed in the API to a *ServiceSearchPattern* attribute. The Salutation Manager then makes an L2CAP connection with a Bluetooth RemDev and sends the *Service SearchPattern* to the RemDev in a `SDP_ServiceSearchRequest`. Note that the *ServiceSearchPattern* may contain `BrowseGroupList` attributes as deemed necessary by the transform process.

If the RemDev Salutation Manager can match existing registered services with the *ServiceSearchPattern*, a `SDP_ServiceSearchResponse` is sent back to the LocDev Salutation Manager containing a list of service record handles for service records that match the *ServiceSearchPattern* in the request.

This process is repeated by establishing an L2CAP connection with another RemDev in radio range of LocDev. When all RemDevs have been contacted in this fashion, the LocDev Salutation Manager builds a list of device IDs returning positive responses and returns them to the calling application as a list of SLM-IDs. The Salutation Manager maintains a list of SLM-IDs and corresponding RemDev addresses for future SDP activity.

Specific transformations from Functional Unit Description Records to a *ServiceSearchPattern* will depend on the definition of *ServiceSearchPattern* for Bluetooth. For example:

LocDev	Client-side Salutation Manager	Bluetooth Service Discovery Protocol	Service-side Salutation Manager	RemDev
			<== slmRegisterCapability() call slmRegisterCapability() return ==>	
	SlmSearchCapability() call ==>			
	SDP_ServiceSearchRequest ==> <== SDP_ServiceSearchResponse : (This step is repeated for each known Bluetooth device. The reply data maybe cached for the next step.)			
	<== slmSearchCapability() return			
	SlmQueryCapability() call ==>			
	SDP_ServiceSearchRequest ==> <== SDP_ServiceSearchResponse			
	SDP_ServiceSearchAttributeRequest ==> <== SDP_ServiceSearchAttributeResponse : (This step is repeated for each service record handle identified in the previous SDP_ServiceSearchRequest.)			
	<== slmQueryCapability() return			
			<== slmUnRegisterCapability() call slmUnRegisterCapability() return ==>	

Table 4-2: Remote Service Discovery Flow Diagram

if the following Service Classes are defined:

DuplexColorPostscriptPrinterServiceClass,
 ColorPostscriptPrinterServiceClass,
 PostscriptPrinterServiceClass,
 PrinterServiceClass,

and the slmSerchCapability() call includes a [Print] Functional Unit Description Record with attributes of Postscript ,Duplex, and Colate,

then the resulting ServiceSearchPattern would contain attributes for PrinterServiceClass and PostscriptPrinterServiceClass.

The TM of the Salutation Manager on RevDev makes a reverse transformation from `ServiceSearchPattern` and compares to the registered Functional Unit attributes.

4.3.2.2 Capability Query

The Salutation `slmQueryCapability()` API call is mapped by the Salutation Manager to a Bluetooth `ServiceSearch` and `ServiceAttribute` protocols in a two step process.

1. The LocDev Salutation Manager's TM maps the `Service Description Record` passed in the API to a `ServiceSearchPattern` attribute. The Salutation Manager then makes an L2CAP connection with the Bluetooth RemDev represented by the SLM-ID passed in the API call, and sends the `ServiceSearchPattern` to the RemDev in a `SDP_ServiceSearchRequest`. Note that the `ServiceSearchPattern` may contain `BrowseGroupList` attributes as deemed necessary by the transform process.

If the RemDev Salutation Manager can match existing registered services with the `ServiceSearchPattern`, a `SDP_ServiceSearchResponse` is sent back to the LocDev Salutation Manager containing a list of service record handles for service records that match the `ServiceSearchPattern` in the request.

2. To determining attribute specifics, the LocDev Salutation Manager selects one of the service record handles returned in Step 1. The Salutation Manager's TM maps the `Service Description Record` passed in the API to an `AttributeIDList` attribute. The Salutation Manager then sends the service record handle and the `AttributeIDList` to the RemDev in a `SDP_ServiceAttributeRequest`.

The RemDev returns a `SDP_ServiceAttributeResponse` containing an `AttributeList` identifying a list of attributes and their values for the requested service record.

Step 2 is repeated for each service record handle returned in Step 1.

When this cycle is completed, the LocDev Salutation Manager assembles a `Service Description Record` from the values returned by in the `SDP_ServiceAttributeResponses`. This `Service Description Record` is return to the calling application.

As before, specific transformations from the `Functional Unit Description Records` (contained in the `Service Description Record`) to a `ServiceSearchPattern` will depend on the definition of `ServiceSearchPattern` for Bluetooth. The same applies to transformations from `Functional Unit Description Records` to `AttributeIDList`. For example:

if the following Service Classes are defined:

DuplexColorPostscriptPrinterServiceClass,
ColorPostscriptPrinterServiceClass,
PostscriptPrinterServiceClass,
PrinterServiceClass,

and the `slmQueryCapability()` call includes a [Print] Functional Unit Description Record with attributes of Postscript, Duplex, and Collate

then the resulting `ServiceSearchPattern` would contain attributes for `PrinterServiceClass` and `PostscriptPrinterServiceClass`, and the `AttributeIDList` would contain attribute IDs for Postscript, Duplex and Collate.

4.3.3 Stop Rules

Instances of the Salutation Manager, such as IBM's Salutation Manager Toolkit, provide a control interface outside of the Salutation APIs. This mapping assumes that such a control interface exists for the Salutation Manager supporting Bluetooth. The Bluetooth stop rules will be set via this interface.

4.3.4 Summary

The Salutation Manager mapping provides a means to use the Salutation Manager as a service broker in the Bluetooth environment. Because Salutation Manager is independent of underlying protocols and operating environments, a Salutation implementation can be a single application interface to numerous protocols. For example, in additions to the Bluetooth mapping, Salutation has been specified for TCP/IP and IR. A mapping to SLP is also being described.

As an example, Figure 4.3 shows a single service discovery application using the Salutation APIs to access the Salutation Manager to locate service in both the Bluetooth and TCP/IP environments. The advantage of this technique is to present a single methodology and API set to application for service discovery. The application need not know where a service resides, and therefore what service discovery primitives to uses, prior to performing a service search.

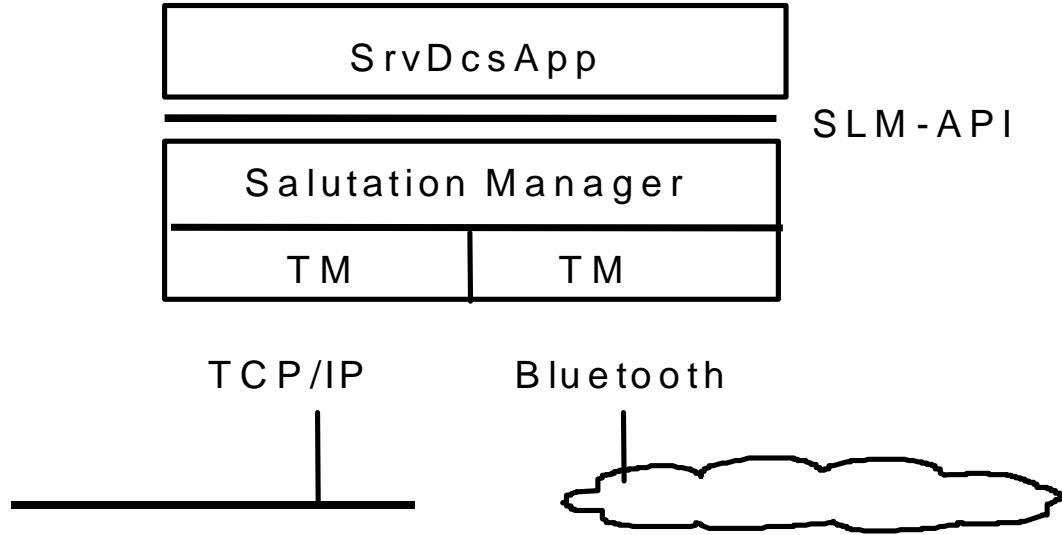


Figure 4.3: Salutation Manager in mixed transport environment

5 References

- [1] *Service Discovery Protocol*, Release 1.0 July 1999
- [2] *Service Discovery Profile*, Release 1.0 July 1999
- [3] *Salutation Architecture Specification (Part 1) Version 2.0c*
(www.salutation.org)

6 Definitions

Term	Definition
Functional Unit Description Record	The Salutation Architecture defines the Functional Unit Description Record as a record that identifies the Functional Unit, and the capabilities of that instance of the Functional Unit. The Functional Unit maps to the Bluetooth Service and capabilities map to the Bluetooth Attributes.
Callback Entry for Open Service Indication	The specified entry of the calling Client is called back by the Salutation Manager when an <i>Open Service</i> request for the Functional Unit is received.
Callback Entry for Close Service Indication	The specified entry of the calling Client is called back by the Salutation Manager when a <i>Close Service</i> request for the Functional Unit is received.
Callback Entry for Receive Data Indication	The specified entry of the calling Client is called back by the Salutation Manager when a <i>Transfer Data</i> request for the Functional Unit is received.
Preferred Functional Unit Handle	If the calling Client wants to be assigned any specific value as its Functional Unit Handle, the preferred value is specified in this parameter. Otherwise, zero (0) should be specified.
Functional Unit Handle	The Salutation Manager generates a unique Functional Unit Handle value and returns it to the calling Client. If the function fails, zero (0) is returned.
SLM-ID	This parameter shall be NULL, indicating the local Salutation Manager, under version 2.0 of the Salutation Architecture.
Service Description Record	The Salutation Architecture defines a Service Description Record as a collection of one or more Functional Unit Description Records. The Service Description Record describes all the services sought by a Client or all the services maintained by a Service.