Bionic Buffalo Tech Note #55

DSM-CC Protocols and Interfaces

Abstract

This *Tech Note* briefly describes the protocols and interfaces defined by DSM-CC (ISO/IEC 13818-6).

Publisher's information and notes are at the end of this document.

Contents

Abstract 1
Contents 1
2. Introduction 1
3. User-Network Configuration Protocol 5
4. User-Network Session Protocol 6
5. User-Network Download Protocol 9
6. User-Network Channel Change Protocol 9
7. User-Network Pass-Through Protocol 10
8. GIOP RPC ("User-User") Protocol 11
9. Objects to Be Implemented on the Client 13
10. Objects to Be Implemented on the Server 15
11. Objects to Be Implemented on Either the Client, or the Server, or Both 16
12. Abstract Interfaces 17
13. Encapsulation of DSM-CC Protocols 18
14. DSM-CC Extensibility and Variations 18
Bibliography and References 19
Publisher's Information and Notes 19

2. Introduction

Note: The DSM-CC specification is not formalized in object terms. In order to employ an object framework, to clarify this discussion, and to summarize a lot of information into a short discussion, this

description uses some terms which are not found in the specification. Although they are not formally defined in the specification, their use is not inconsistent with the specification.

DSM-CC describes an architecture of multimedia service providers and service consumers, as well as a network management component within the network itself. The network management component is called a *session and resource manager*, or SRM. Each of these components plays multiple roles.

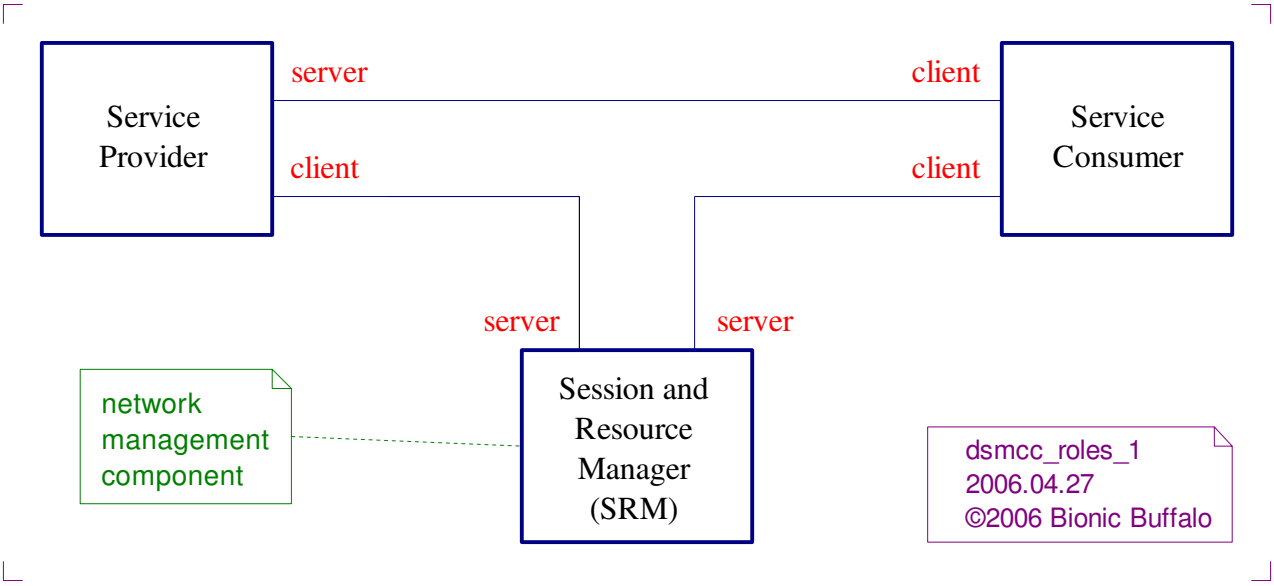


Illustration 1: Basic roles in a DSM-CC system

In other words, a service provider is both a client and a server, while a service consumer is a client in two different relationships. In DSM-CC terms, the service provider and service consumer are both *users* of the network. In this context, DSM-CC defines six different protocols, some of which are termed "user-network" but are in fact "user-user". The two U-N protocols which are in fact U-U protocols are probably so-named because they use the same headers as the three "true" U-N protocols.

The six protocols are:

- the U-N *configuration* protocol, with which a service provider or service consumer communicates with the SRM to obtain configuration information
- the U-N *session protocol*, with which a service provider or service consumer communicates with the SRM to create sessions and to acquire the use of resources
- the U-N *download* protocol (in fact, a user-user protocol), with which a download client acquires files or objects from a download server
- the U-N *channel change* protocol (in fact, a user-user protocol), with which a *switched digital broadcast* (SDB) client requests channel changes from a SDB server

- the U-N *pass-through* protocol, with which one network user (server or client) utilizes the SRM to pass messages to, or receive messages from, another network user
- the *remote procedure call*, or RPC, protocol, often called the *User-User* protocol, with which a service consumer communicates directly with a service provider, to make requests and to receive responses; this protocol is a dialect of the CORBA *general inter-ORB protocol* (GIOP)

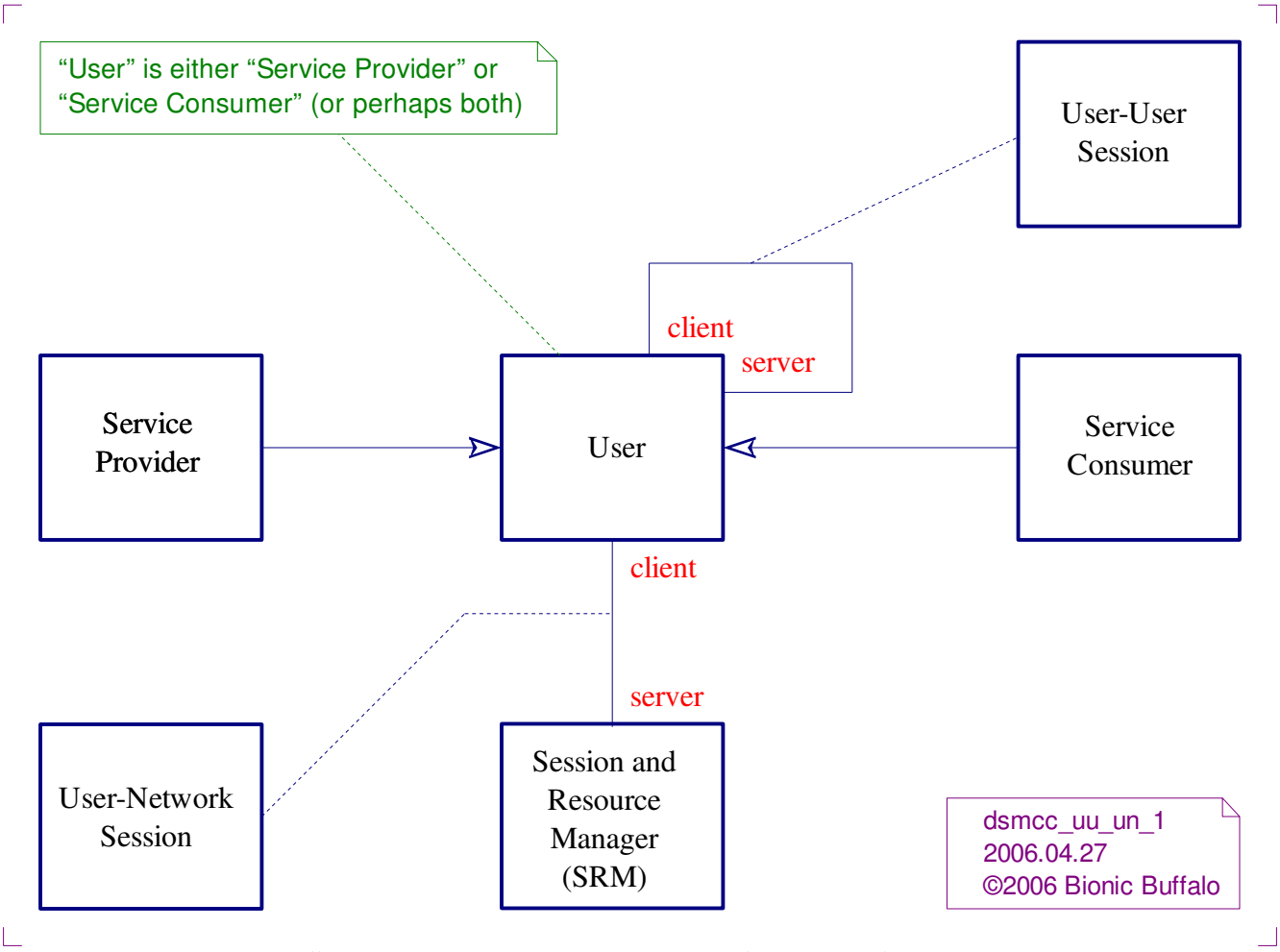


Illustration 2: Participants in user-user and user-network sessions

In addition to these protocols, the DSM-CC specification defines various objects and their programming interfaces. Some of these objects to be implemented on the content provider (server), while other objects are to be implemented on the content consumer (client), and still others might be implemented on either machine.

The architectural model used for object implementation is CORBA. Objects are never directly accessed by client applications, but always employ an *object request broker* (ORB) as an intermediary.

When the object is on the same machine as the client application, then the ORB is simply interposed into the client-server relationship. The API seen by the client application is defined by a mapping from the object's interface definition language (IDL) to the programming language used by the application.

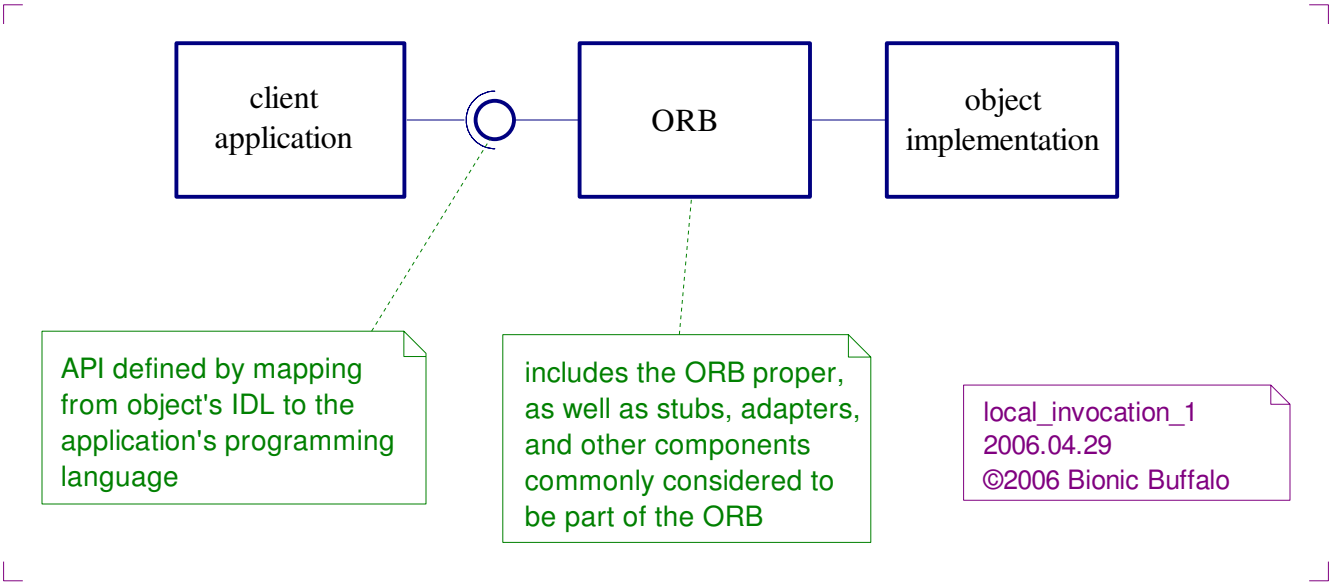


Illustration 3: Invocation of method on local object

On the other hand, when the client application and object implementation are on different machines, then two ORBs are employed, one on the client machine and one on the server (object implementation) machine. The ORBs function together as a single intermediary, using the GIOP RPC protocol between them. The API seen by the client application is the same as when a single ORB is used to communicate with a local object implementation.

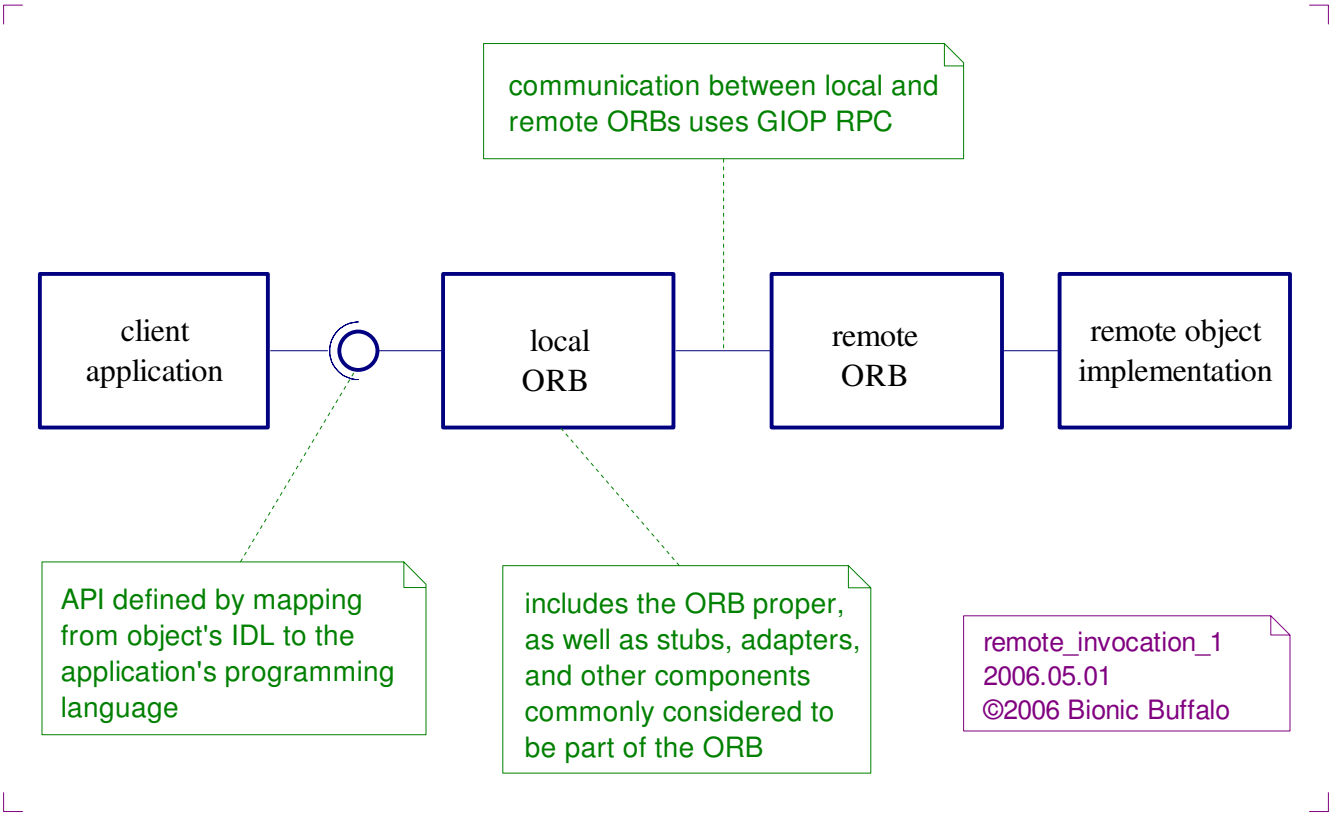


Illustration 4: Invocation of method on remote object

The GIOP RPC protocol is the over-the-wire protocol among ORBs. It is not used in the download or channel change protocols, but it is used when a client machine communicates with one of the objects defined in IDL by DSM-CC, implemented on the server machine.

(*Note:* DSM-CC does not explicitly require an ORB. However, it requires much of the same functionality. When many of those functions are aggregated, they might as well be called an ORB, which is what this discussion does. As a practical matter, if a pre-built or separate ORB is not used, then an implementor will have constructed most of one before the system is completed.)

(*Note:* DSM-CC was specified during earlier days of CORBA. The general architecture has not changed, but some implementation details in the DSM-CC specification still reflect the earlier CORBA standards. This discussion updates some of DSM-CC to reflect current CORBA implementation details.)

3. User-Network Configuration Protocol

The user-network configuration protocol allows dynamically configured network users to acquire configuration information from the network. Unless all configuration information is statically programmed into the user devices, a user or the network will use this before any of the other DSM-CC

protocols. It may be used to configure both service providers and service consumers. Configuration may be requested by the user, or it may be sent without request by the network.

The user-network configuration protocol is based on any lower level protocol supporting datagrams. Depending on how it is used, it may employ a one-way connection (from network to user), or a two-way connection.

DSM-CC does not define specific configuration parameters. However, the kinds of parameters envisioned for use include such things as client or server network addresses, session parameters (such as timeout parameters and retry count limits), protocol or interface version identifiers, and service profile identifiers.

Note that a user device might be connected at one time or another to more than one physical network, or there might be multiple logical networks accessible from a single physical network connection. For example, a video on demand server might support the user-user protocol over TCP/IP at one address, while download services might be offered either at some different IP address or within an MPEG data stream. The user-network configuration protocol allows a user device (server or client) to select the parameters (including, perhaps, the network and protocol) needed to establish sessions with other devices.

4. User-Network Session Protocol

The user-network session protocol allows the network to establish sessions among users, and to assign resources to sessions. The establishment of sessions and the assignment of resources may be done at the behest of the network, or they may be requested by users.

The user-network configuration protocol is based on any lower level protocol supporting datagrams or stream connections. Depending on how it is used, it may employ a one-way datagram connection (from network to user), or a two-way datagram or stream connection. The one-way mechanism severely limits the functionality. A two-way connection is almost invariably used.

Sessions are asymmetric, each having a client and a server. Within the network, the session is identified by a session identifier. Each user (client or server) is identified by an identifier, the server identifier or client identifier. User identifiers are open systems interconnection (OSI) network service point (NSAP) addresses.

The session protocol includes message sequences to set up new sessions, release sessions, inquire about or report on session status, and to transfer a session from one server to another server. The sequences may be initiated by the network, or by the user.

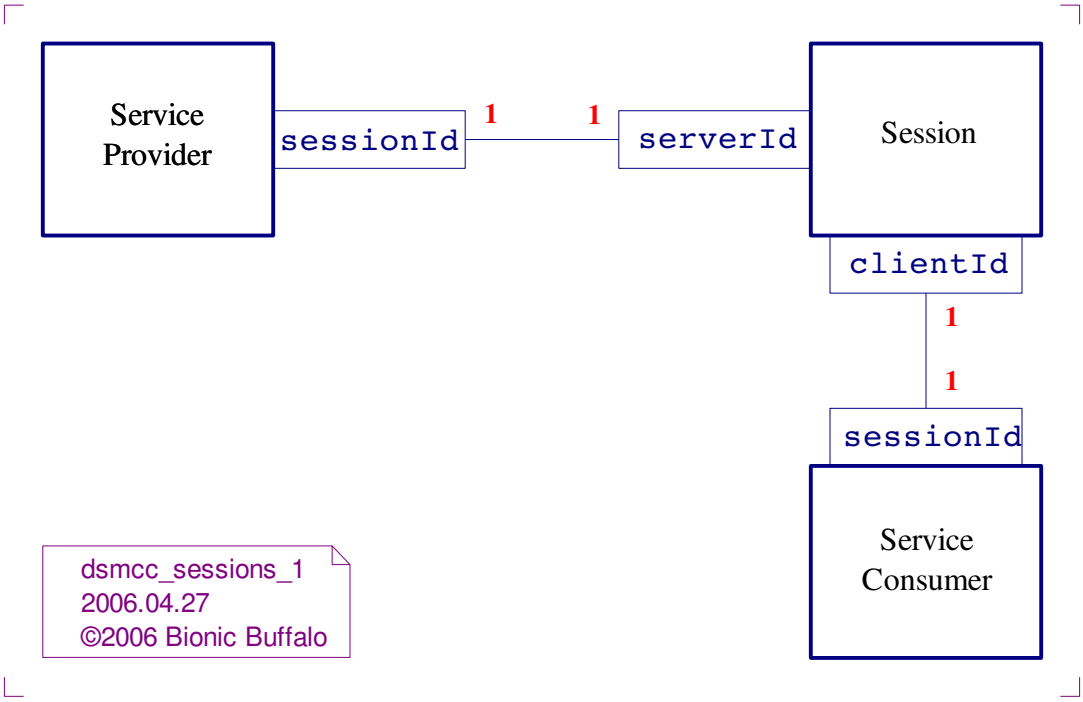
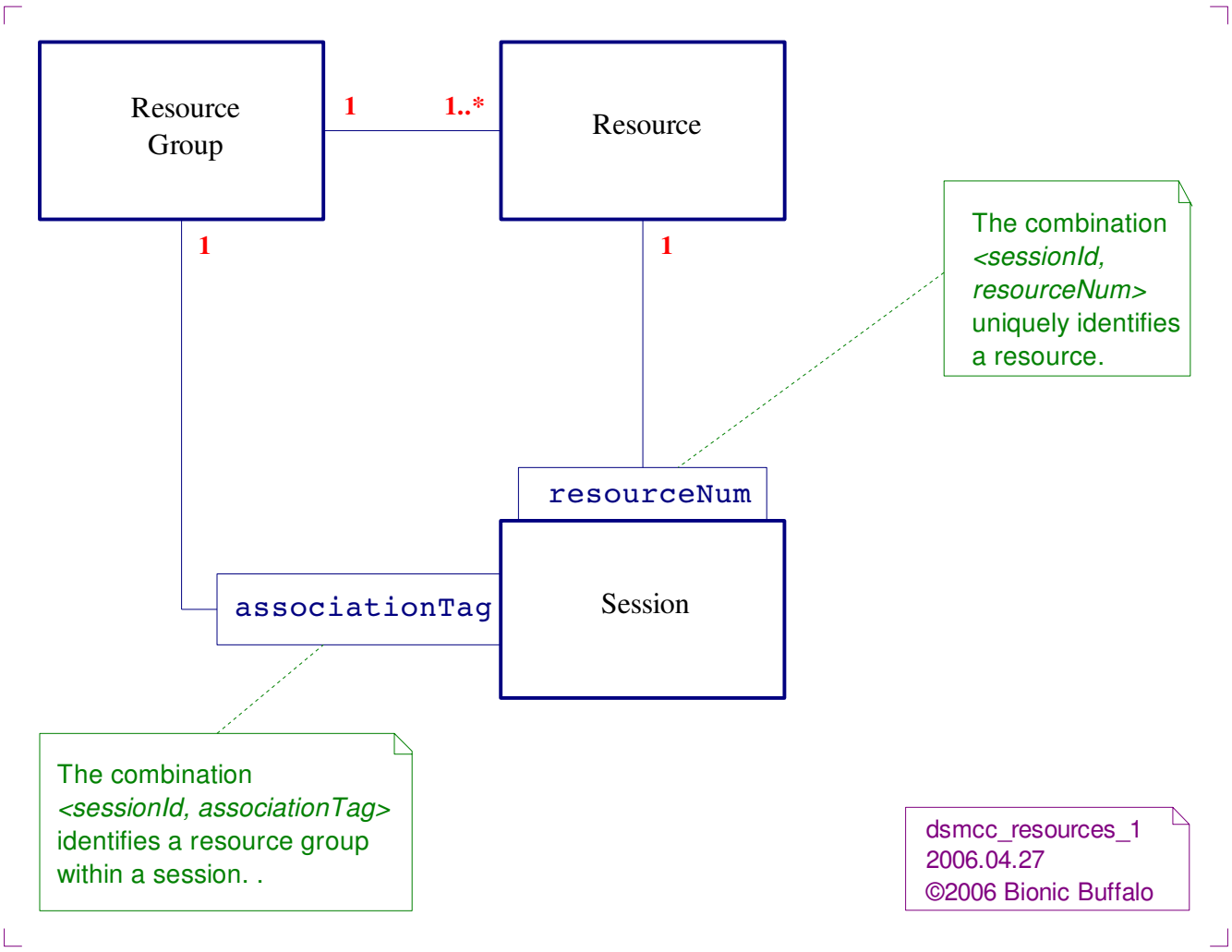


Illustration 5: Session, server, and client identifiers

The session protocol includes message sequences to add resources to a session, to delete resources, and to inquire about or to report status. Sequences may be initiated by the network, or by the user.

Some types of resources are defined by the specification, but the list is extensible by the implementor. The kinds of resources defined by DSM-CC include various kinds of connections (ATM, ISDN, and so on), physical channels, MPEG programs, broadcast feeds, and others.

Resources may be shared among sessions, and a single user may participate in more than one session.



Resources are numbered within a session, using `resourceNum`. Resources may be grouped; a group of associations is denoted by an `associationTag`.

The address of a resource as seen at one point in the network may be different from the address of the same resource as seen at a different point. This happens because of network address translation (NAT), multiplexing and demultiplexing of MPEG streams, channel mapping, frequency shifting, and other operations. One of the functions of the SRM is to notify client and server users of the addresses of resources, as seen at the point of the client or server. Therefore, the address or channel given to the server may be different from the address or channel given to the client. This make passing addresses between server and client problematic. To avoid this problem, when it is necessary to make reference to a channel or address, the client or server will instead use the `associationTag`, which is the same at both ends of a session.

5. User-Network Download Protocol

The download protocol is a simple protocol used to transfer files, data, or objects from server to client. There are three variations (called *scenarios*), sharing the same message formats but operating differently:

- *flow-controlled* download is similar to ftp or other one-on-one file transfer protocols, using flow-control, requests for specific blocks, and sliding windows
- *non-flow-controlled* download omits requests for specific blocks; a server can transfer to multiple clients at once
- *carousel* download involves repeated transmissions from the server, independent of client requests; a specialized form, *object carousels*, defines the encapsulation of certain types of DSM-CC objects within the data

When download uses two-way communication, the channels often are asymmetric, with the data sent in a high speed channel such as an MPEG data stream. In the carousel scenario, it is often used to send data common to all clients, as electronic program guides, stock tickers, weather information, or software updates.

With object carousels, copies of the objects are sent to the client, and the methods for those objects are implemented on the client. Therefore, the object copies appear to be local objects, although they originated on the server.

Of all the DSM-CC protocols, download (especially carousel download) is the most common. Often carousels are used when no other DSM-CC protocols are used. The download portion of DSM-CC has been incorporated into other specifications.

6. User-Network Channel Change Protocol

The channel change protocol is defined to allow a client to change channels remotely in a *switched digital broadcast* (SDB) environment. The SDB client makes channel change requests to an SDB server, which is part of an *interworking unit* (IWU).

CCP and IWUs exist in environments where the IWU can receive more channels than the client. For example, a great number of channels might be brought into an IWU in a neighbourhood using a high bandwidth connection, while individual clients or homes in the neighbourhood are capable of receiving only a few channels at a time. CCP can be used to select from the channels available at the IWU.

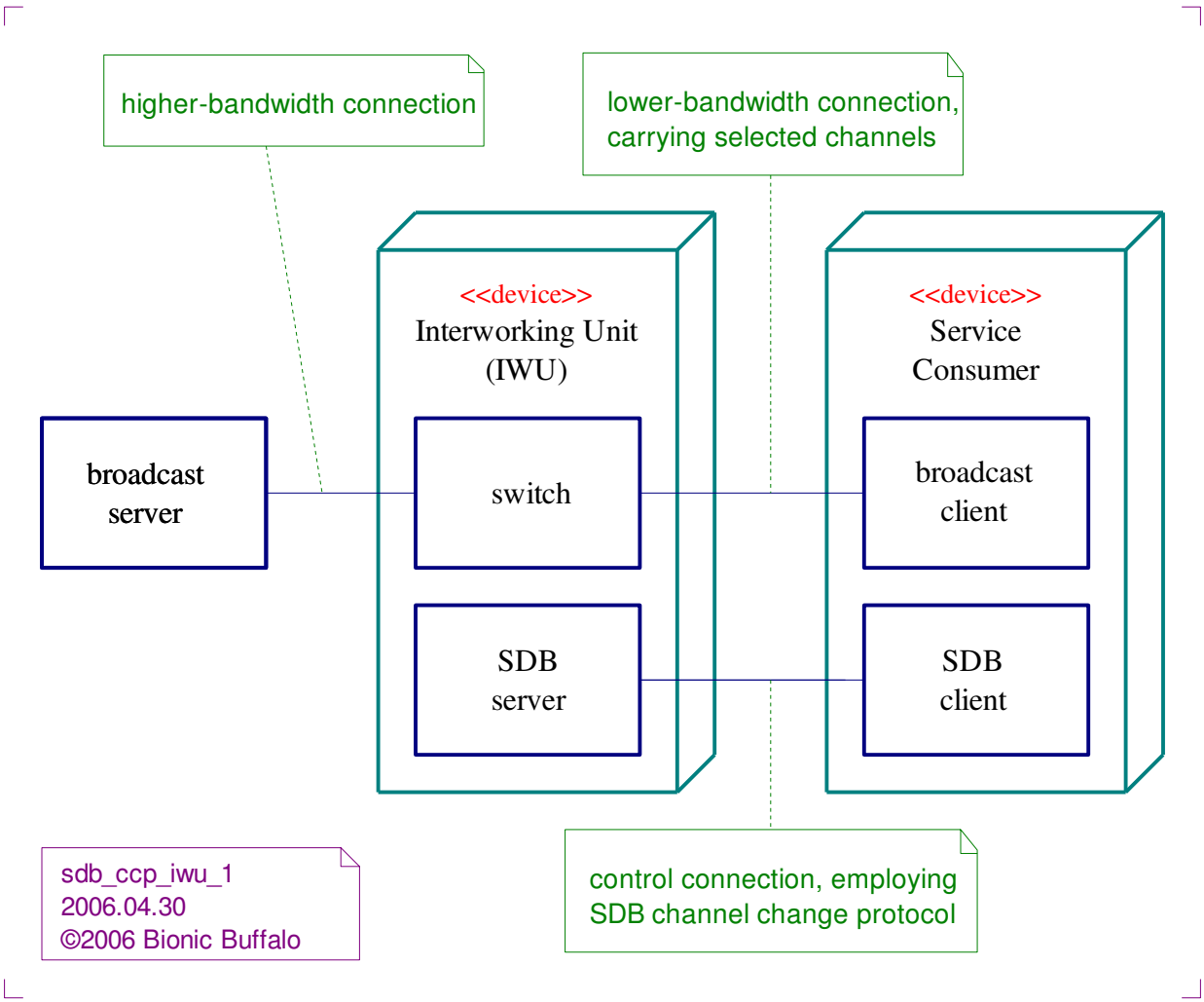


Illustration 6: The SDB channel change protocol between an IWU and a broadcast client

In a pure DSM-CC environment, the IWU in the diagram above participates in two sessions: one with the broadcast server, and another with the SDB client.

There are also various non-DSM-CC, proprietary protocols similar to the channel change protocol.

7. User-Network Pass-Through Protocol

The pass-through (pass-thru) protocol is used to send messages from one user (server or client) to another user (server or client), using the network as an intermediary. It includes the option for the recipient to return a responsive message. Message payloads are arbitrary octet sequences.

8. GIOP RPC ("User-User") Protocol

CORBA defines the *general inter-ORB protocol* (GIOP) for use among ORBs. Within the messages, certain fields are user-extensible. DSM-CC uses these extensible fields to define a dialect of GIOP. The fundamental structure of GIOP is not altered. DSM-CC IIOP retains the same message syntax as CORBA GIOP, with additional semantics. The protocol state machines for DSM-CC GIOP and CORBA GIOP are the same.

A standard CORBA ORB which supports the *portable interceptor* (PI) interface can support DSM-CC GIOP if the application includes handlers for the necessary extensions.

The DSM-CC specification does not in fact describe how GIOP works: it describes only the extensions for DSM-CC. For that reason, and also because GIOP is well described elsewhere, this discussion will omit a detailed description of GIOP. Instead, this discussion will focus on the primary extensions defined for GIOP by DSM-CC.

A critical element of GIOP is the *interoperable object reference*, or IOR. An IOR is used to identify an object uniquely, at the same time providing information necessary to locate the object. The information needed to use a given protocol to access the object is contained within a *profile*, within the IOR.

In a simple network, there might be one profile within an object's IOR. For example, the IOR for an object in a simple internet application might include the IP address, port number, and a unique identifying key to distinguish the object from other objects at the same port and address.

In a more complex system, there might be more than one way to access the same object. For instance, in a DSM-CC environment, the object might be accessible through the internet, by way of an MPEG stream, and also on a carousel broadcast on some channel. In the case of a `DSM::Stream` object, which encapsulates video or audio content, the content itself might be accessed using a high speed MPEG stream, while playback control (pause, play, rewind, and so on) might best be done using a lower speed IP connection. Multiple protocols can share a profile, or there might be one profile for each protocol. (A single protocol is not permitted to use more than one profile.)

In addition to the basic addressing information, each profile might contain additional information used to access the object. Known as components, these extra profile data provide protocol parameters; security, transaction processing, and routing mechanisms, policies, and parameters; and other useful items.

DSM-CC defines a structure `DSM::Tap` which contains connection information for various types of connection. An instance of `DSM::Tap` contains four members:

- the tap *identifier*, which is used to identify or number the tap
- the tap *use*, which says how the connection is employed (such as MPEG transport downstream,

download control, RPC, SDB control)

- the `associationTag` from the user-network resource; given this, the network address or channel can be determined
- a *selector*, which is used when multiplexing a connection, and is used to choose which data in the connection is associated with this tap

A sequence of `DSM::Tap` is a `DSM::ConnBinder`, and it contains the information, or references to the information, necessary to access an object or service.

CORBA defines a profile and components for internet protocol connections, but allows additional profiles and components to be defined for other protocols. DSM-CC defines eight such new profiles:

- the *minimum* profile, which contains a `DSM::ConnBinder` as a component
- the *child* profile, which contains a `DSM::ConnBinder` and a name string; this is intended for use in references to `DSM::Composite` objects, described in a subsequent section
- the *option* profile, which can contain any of several components, as needed: a `ConnBinder`, an internet address with port number, a name string, an interface code, an object key, and a service location
- three “lite” (light) profiles, *lite minimum*, *lite child*, and *lite option*; these are the same as the first three, but with different encoding to result in smaller messages
- the *BIOP* profile (for “broadcast inter-ORB protocol”), which contains a component with information to identify an object within a carousel
- the *ONC* profile, which contains, in addition to other possible component, a program number and version associated with an *open network computing* (ONC) service; ONC defines an RPC mechanism in wide use prior to CORBA

In addition to the profiles and their components, CORBA also defines a number of `ServiceContext` parameters. These parameters are included with GIOP request and reply messages. They provide additional information to the server or client to be used when handling the request. CORBA allows the definition of additional `ServiceContext` parameters; DSM-CC has defined six additional such parameters:

- `DSM::CompatibilityDescriptor` describes the hardware and software being used
- `DSM::DownloadInfoRequest` and `DSM::DownloadInfoResponse` are used to negotiate parameters involved in collateral download operations
- `DSM::AuthRequest_T` conveys information required for a security authentication procedure

- `DSM::ConnBinder` caries tap information needed when file transfer or `read()` operations involves a secondary channel (as when using download)
- `DSM::Version` selects from among several available versions in a `resolve()` operation

It is important to note that, under CORBA, a client might be transferred to a new or different server while a request is outstanding. GIOP supports such service transfers. This is a crucial feature when constructing high-availability service and content delivery systems.

9. Objects to Be Implemented on the Client

Objects with certain interfaces are expected to be implemented only on the client (service consumer) device. RPC invocations by server (service provider) applications are not anticipated. Invocations by client applications involve the local ORB only, and do not require the network. This section describes those client objects.

`DSM::Session` is created in some proprietary fashion, and the way an application acquires a reference to it is undefined by DSM-CC. Once the application has the reference, it calls `attach()` to connect to a server, and `detach()` to disconnect. The `attach()` operation takes as input various parameters describing the server and path to the service, and returns object references to a `ServiceGateway` and possibly to an initial service object.

Sessions can be suspended and resumed. If requested by a parameter, the `detach()` operation will suspend as session, rather than ending it. If requested by a parameter, the `attach()` operation will resume a suspended session, rather than starting a new session. It is possible to have multiple active sessions at once, with the same or with different `ServiceGateway`s.

The `attach()` process takes place in one of two ways, depending on whether the U-N session protocol is used.

- If the U-N session protocol is used, then `DSM::Session::attach()` uses a local `DSM::SessionUU` object to establish a U-N session. The resolved object references are returned by `DSM::SessionUU::attach()`. A `ServiceGateway` is resolved to a remote `DSM::ServiceGatewayUU` object.
- If the U-N session protocol is not used, then `DSM::Session::attach()` invokes a remote `DSM::SessionSI::attach()` operation on the server, which returns the resolved object references via the network. A `ServiceGateway` is resolved to a remote `DSM::ServiceGatewaySI` object.

In either case, `DSM::Session::attach()` does not return the resolved `ServiceGatewayUU` or `ServiceGatewaySI` reference. Instead, it returns a reference to a local `DSM::ServiceGateway`

object. The application sees only the local `ServiceGateway` object, not the remote object.

If requested by the server, `attach()` may also initiate a download of objects required for the session to begin.

Similarly, `DSM::Session::detach()` invokes either the local `DSM::SessionUU::detach()` operation, or the remote D`SM::SessionSI::detach()` operation.

`DSM::SessionUU` is used by the `DSM::Session` object to create a U-N session with a server. This interface is not used if the U-N session protocol is not used. In any case, it is not used by an application. The `SessionUU` object implements the U-N session protocol. Resolved object references are returned within the `UserData` field of the server response message, and passed back to the calling `Session` object.

`DSM::ServiceGateway` inherits both `DSM::Directory` and `DSM::Session`. In other words, it appears to be a directory of services which also can establish sessions. Although the `ServiceGateway` object is local, the directory belongs to a remote server.

If the U-N session protocol is used, then the `attach()` and `detach()` operations are performed with the assistance of a local `SessionUU` object, and the directory operations are implemented with calls to a remote `ServiceGatewayUU` object.

If the U-N session protocol is not used, then all operations are implemented by calls to a remote `ServiceGatewaySI` object.

`DSM::Download` implements a portion of the client side of the download protocol. Although an object which has or inherits the `Download` interface will be defined on the server, the `DSM::Download` interface itself will be implemented on the client. The server will implement the `DSM::DownloadSI` interface, which is not used by the application.

The local `Download` object has a high-level interface to the application, offering four operations:

`info()` lists the available modules, along with information about them

`alloc()` allocates buffers at the client

`start()` initiates a download of selected modules

`cancel()` terminates a download operation in progress

The remote `DownloadSI` object implements a lower level interface. Control information is sent over the RPC channel, while the data blocks are usually sent over a higher speed, one-way channel. The application does not see the lower level protocol.

10. Objects to Be Implemented on the Server

Objects with certain interfaces are expected to be implemented on the server (service provider) device. Client (service consumer) applications at a remote device invoke methods on these objects using the remote ORB, which along with the server ORB and an intervening GIOP connection, mediates the requests. It is possible, of course, for server applications to invoke operations on those server objects locally, without involving the network. This section describes those server objects.

`DSM::SessionSI` is used when the U-N session protocol is not used. It provides the `attach()` and `detach()` operations required by a client implementation of `DSM::Session`.

`DSM::SessionSI::attach()` returns the `DSM::ServiceGatewaySI` reference, and possibly a reference to a first service, and may indicate that certain objects must be downloaded for the session to begin.

`DSM::ServiceGatewayUU` provides remote directory operations to a client `DSM::ServiceGateway` object when the U-N session protocol is employed.

`DSM::ServiceGatewaySI` provides remote directory operations, as well as `attach()` and `detach()`, to a client `DSM::ServiceGateway` object when the U-N session protocol is not employed.

`DSM::DownloadSI` implements the server end of the download protocol, with RPC being used to carry the control information.

`DSM::View` is an alternative interface for a directory or database. It allows queries on the content using a minimal subset of SQL.

`DSM::Interfaces` is a service used to manage the interfaces of objects in a DSM-CC system. It is a kind of management front-end for the interface repository defined by CORBA. Four operations are defined on this interface:

`show()` returns certain information about an interface, including the IDL defining the interface

`define()` defines new interfaces and type definitions

`check()` verifies the coherency of the interface repository

`undefine()` removes a definition from the repository

Unlike standard CORBA, DSM-CC assigns numeric identifiers to all interface and type definitions in the repository. The `DSM::Interfaces` interface supports the use of these numeric identifiers.

11. Objects to Be Implemented on Either the Client, or the Server, or Both

Objects with certain interfaces might be implemented on the client or on the server, depending on the context and application. When implemented on the client, there is no expectation of invocation by any server application. This section describes those client or server objects.

`DSM::Directory` is an extension of the standard `CosNaming::NamingContext` service used in most CORBA environments. As `CosNaming` is described in depth elsewhere, it will not be discussed much here. This discussion will be limited mostly to the extensions.

`DSM::Directory` will be implemented on the client for local client objects (as, for instance, if the client has a local filesystem), and for objects downloaded from an object carousel. It will be implemented on the server for objects residing on the server.

`DSM::Directory` defines a `PathSpec` structure, which is different from a `CosNaming::Name`. While a `Name` represents one object (as with *"colour/red"*), a `PathSpec` may represent multiple objects. Although there is no defined string representation for a `PathSpec`, we might imagine a representation such as *"colour/red,green,blue"* to represent three objects named *red*, *green*, and *blue*.

`CosNaming` defines various operations on a `NamingContext` object. A `NamingContext` corresponds roughly to a directory of objects, with each object bound to a name. `NamingContext`s, as objects themselves, may be bound to names in other `NamingContext`s, resulting in directory trees. In addition to the operations inherited from `CosNaming::NamingContext`, `DSM::Directory` has four additional operations:

`open()` resolves the (possibly multiple) objects associated with a `PathSpec`

`close()` deletes the resources associated with a reference to the `Directory`

`get()` returns attribute values associated with the objects associated with a `PathSpec`

`set()` sets attribute values associated with the objects associated with a `PathSpec`

`DSM::File` represents an ordinary file, with `read()` and `write()` operations. It will be implemented on the client for local client files and for files downloaded from an object carousel. It will be implemented on the server for files residing on the server.

`DSM::Stream` represents streaming audio or video content. It includes methods to control playback: `pause()`, `resume()`, `status()`, `reset()`, `jump()`, and `play()`.

A `Stream` object in a server is implemented on the server. The client must, of course, implement code to handle the incoming data, but the `DSM::Stream` operations are implemented on the server object. A single instance of a `Stream` object may handle multiple simultaneous client connections. (This is unlike some non-DSM-CC architectures which replicate a content object for each user.)

When a `Stream` object is downloaded to the client from a carousel, the downloaded object is implemented on the client, but supports only a limited subset of the defined operations. The downloaded object state consists of references (taps) describing the network address of a broadcast program. No interactive playback support is provided.

12. Abstract Interfaces

Some interfaces described in DSM-CC are abstract and not instantiable. In other words, they can be inherited by other DSM-CC interfaces, but an object with such an interface alone never exists.

`DSM::Access` defines attributes which provides size, update time and date, lock status, and a mask of which user groups have manager, broker, writer, or reader access to an object. Most instantiable DSM-CC interfaces inherit `DSM::Access`.

`DSM::Base` provides two operations: `destroy()` and `close()`, for the destruction of objects and for the release of references. Most instantiable DSM-CC interfaces inherit `DSM::Base`.

`DSM::Composite` allows a set of child objects to be associated with a given (parent) object; the parent object inherits the `DSM::Composite` interface. A version is associated with each child object. The interface exists for the convenience of developers, and is not required by any part of the DSM-CC specification.

`DSM::Config` is used to implement a form of asynchronous operation call on the object which inherits the `Config` interface. An attribute `DeferredSync` is used to switch the operation calls between synchronous and asynchronous mode. In asynchronous mode, calls immediately return a request handle from the stub, without waiting for the operation to complete. With the `inquire()` operation, the request handle may be used to check status. With the `wait()` operation, the handle may be used later to wait for operation completion.

`DSM::Event` may be inherited by DSM::Stream. It is by an application to intercept event information embedded in the content stream. The `subscribe()` operation subscribes to an event, while `unsubscribe()` cancels a subscription. The `notify()` operation blocks, not returning until one of the subscribed events has been received.

`DSM::First` may be inherited by a `DSM::Session` object. It provides two operations: `root()` to obtain the `ServiceGateway` reference, and `service()` to obtain a reference to the primary service object.

`DSM::Kind` may be inherited by any instantiable interface. It provides two operations: `has_a()` to determine if an object supports a given interface, and `is_a()` to list the interfaces supported by an object.

`DSM::LifeCycle` may be inherited by any instantiable interface. It has a single operation, `create()`, which returns a persistent, interoperable object reference (IOR) for the object upon which the operation is called.

`DSM::Security` may be inherited by any instantiable interface. It has a single operation, `authenticate()`, which allows a client to provide credentials which might be necessary to gain access to the target object.

`DSM::State` may be inherited by any instantiable interface. It includes two operations: `suspend()` requests that the object return its state with respect to the client and suspend operations, and `resume()` requests that the object resume operations using the given state.

13. Encapsulation of DSM-CC Protocols

The five U-N protocols (configuration, session, download, channel change, and pass-through) may be positioned above some other datagram transport protocol. Minimum requirements for the transport protocol are:

- error detection must be provided (as with a CRC remainder or similar mechanism)
- messages with errors may simply be discarded, and lost message detection is not necessary
- no control over the rate of transmission is required
- if messages are fragmented by the transport layer during transmission, then they must be reassembled by the transport layer
- messages may be delivered out of sequence

The RPC (by default, GIOP) protocol used determines the necessary characteristics of the transport layer required for the RPC protocol. Typically, this is a reliable, guaranteed transport such as TCP/IP.

Any or all DSM-CC protocols may be encapsulated within MPEG-2 transport streams. The specification details the mechanisms to be used in that case. A description of such encapsulation is beyond the scope of this *Tech Note*.

14. DSM-CC Extensibility and Variations

All DSM-CC message formats and protocols allow considerable extensibility.

The five U-N protocols employ these main mechanisms:

- the messages include private data areas, whose syntax and semantics are left to the implementor

- many data structures (such as taps, compatiblity descriptors, and resource descriptors) allow implementation specific use subject to some syntactical constraints
- various types and tags include user-assigned values, or allow new values to registered by implementors

The U-U or RPC protocol provides mechanisms defined by the underlying protocol (typically GIOP). In addition, the additional data structures (such as taps, compatibility descriptors, and resource descriptors), most of which are shared with elements of the U-N protocols, offer the same extensibility mechanisms available to U-N protocol implementors.

It should be noted that the extensibility mechanisms can severely hinder interoperability. For example, the version of U-N used by Scientific Atlanta and interoperable vendors make such heavy use of private data instead of using the specification defined mechanisms, that their implementation is for all intents and purposes a proprietary protocol; and their U-U protocol, although it has some aspects of DSM-CC U-U, is almost unrecognizable in terms of the specification.

Bibliography and References

ISO/IEC 13818-6, Information Technology - Generic coding of moving pictures and associated audio information - Part 6: Extensions for DSM-CC, 1998.09.01

OMG formal/02-11-01, CORBA Core Specification 3.0, 2002.11

Publisher's Information and Notes

Publisher. Bionic Buffalo Corporation, 502 North Division Street, Carson City, Nevada 89703, USA.

Current Version. The current version of this document may be found at *http://www.tatanka.com/doc/technote/tn0055.pdf* (for PDF) and *http://www.tatanka.com/doc/technote/tn0055.html* (for HTML).

Security. Unrestricted. There are no security restrictions on the distribution of this document.

Other *Tech Notes*. For available *Tech Notes*, visit the Bionic Buffalo web site at *http://www.tatanka.com/doc/technote/index.htm*, or e-mail *query@tatanka.com*. Most Bionic Buffalo *Tech Notes* are available in both HTML and PDF form.

Key. PGP/GnuPG key fingerprint: `a836 e7b0 24ad 3259 7c38 b384 8804 5520 2c74 1e5a`.