CCSP Message Bus

CCSP Message Bus is used by the CCSP components to communicate with each other and send notifications to registered subscribers in a single processor environment. CCSP message bus uses D-Bus/R-Bus for IPC.

D-Bus

This section is not intended to be a tutorial on D-Bus. However there are a few points that need to be well understood.

In D-Bus, the bus is a central concept. It is the channel through which applications can do the method calls, send signals and listen to signals. There are two predefined buses: the session bus and the system bus.

- The session bus is meant for communication between applications that are connected to the same session
- The system bus is meant for communication when applications (or services) running with disparate sessions wish to communicate with each other. Most common use for this bus is sending system wide notifications when system wide events occur.

Normally only one system bus will exist, but there can be several session buses.

A bus exists in the system in the form of a bus daemon, a process that specializes in passing messages from a process to another. Sending a message using D-Bus will always involve the following steps (under normal conditions):

- Creation and sending of the message to the bus daemon. This will cause at minimum two context switches.
- Processing of the message by the bus daemon and forwarding it to the target process. This will again cause at minimum two context switches.
- The target component will receive the message. Depending on the message type, it will either need to acknowledge it, respond with a reply or ignore it. Acknowledgement or replies will cause further context switches.

In addition to the context switches, described above, the data from the “replies” gets copied when it enters the D-Bus library, copied again as it enters the sockets to the bus, which then sends it into another socket to the client, which then makes a copy. These copies can be really expensive.

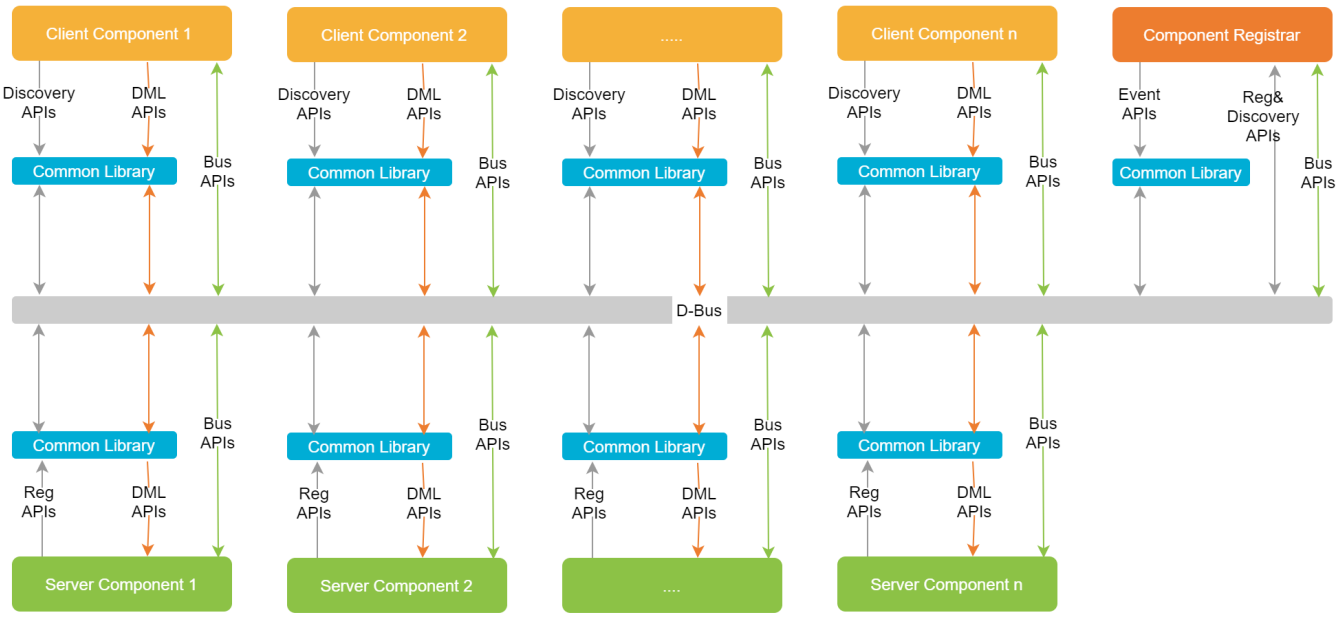
Coupled together, the above rules indicate that D-Bus is not efficient in transferring large amounts of data between processes.

D-Bus, however, provides a useful feature that allows passing UNIX file descriptors over the bus. The idea is to use DBus for flexible IPC method calls between a client and a server and a dedicated pipe for passing large results from the server back to the client. This allows very fast transmission rates, while keeping the comfort of using DBus for inter process method calls. The file descriptor may even point to a shared memory object allocated via shm_open (). Data Plane deals with this in greater detail.

D-Bus Bindings

D-Bus Bindings must be auto generated using a D-Bus binding tool such as dbus-binding-tool from D-Bus-Glib, using the introspection XML file that defines the component’s interface and supported signals. The Adapter bindings should only be thin glue for bridging Component interfaces. In other words, there should not be any component functionality directly implemented in the bindings, other than just calling into the component’s interface.

D-Bus Architecture diagram



D-Bus Remote Communication Capabilities

Applications using D-Bus are either servers or clients. A server listens for incoming connections; a client connects to a server. Once the connection is established, it is a symmetric flow of messages; the client-server distinction only matters when setting up the connection. When using the bus daemon, the bus daemon listens for connections and component initiates a connection to the bus daemon.

A D-Bus address specifies where a server will listen, and where a client will connect. For example, the address

unix:path=/tmp/abcdef

specifies that the server will listen on a Unix domain socket at the path /tmp/abcdef and the client will connect to that socket. An address can also specify TCP/IP sockets. For example the address

tcp:host=10.1.2.4,port=5000,family=ipv4

specifies that server at 10.1.2.4 will listen on tcp port 5000 and the client will connect to that socket. From D-Bus specification at https://dbus.freedesktop.org/doc/dbus-specification.html#transports-tcp-sockets

TCP Sockets

The TCP transport provides TCP/IP based connections between clients located on the same or different hosts. Using TCP transport without any additional secure authentication mechanisms over a network is unsecure.

Server Address Format

TCP/IP socket addresses are identified by the “tcp:” prefix and support the following key/value pairs:

Name	Values	Description
host	(string)	dns name or ip address
port	(number)	The tcp port the server will open. A zero value let the server choose a free port provided from the underlying operating system. libdbus is able to retrieve the real used port from the server.
family	(string)	If set, provide the type of socket family either "ipv4" or "ipv6". If unset, the family is unspecified.

Unix Domain Name sockets are used for local intra processor IPC where as TCP/IP sockets are used for remote inter processor communication.

The D-Bus daemon has a configuration file that specializes it for a particular application. One of configuration parameter of interest is the listen element. From the man page of D-Bus daemon https://dbus.freedesktop.org/doc/dbus-daemon.1.html, the following configuration elements are defined.

<!ELEMENT busconfig (user |

type |
fork |
keep_umask |
listen |
pidfile |
includedir |
servicedir |
servicehelper |
auth |
include |
policy |
limit | selinux)*>

The listen element defines address that the bus should listen on. The address is in the standard D-Bus format that contains a transport name plus possible parameters/options.

Example: <listen>unix:path=/tmp/foo</listen>
Example: <listen>tcp:host=localhost,port=1234</listen>

If there are multiple <listen> elements, then the bus listens on multiple addresses. The bus will pass its address to started services or other interested parties with the last address given in <listen> first. That is, apps will try to connect to the last <listen> address first.

tcp sockets can accept IPv4 addresses, IPv6 addresses or hostnames. If a hostname resolves to multiple addresses, the server will bind to all of them. The family=ipv4 or family=ipv6 options an be used to force it to bind to a subset of addresses
Example: <listen>tcp:host=localhost,port=0,family=ipv4</listen>

R-Bus*

RBus is a 3 layered RPC communication bus.

- At the lowest layer, it is rtMessage, which provides basic messaging capabilities across Unix domain or TCP sockets.
- Above that is "rbus-core", which is an intermediate layer that offers RPC and eventing capabilities.
- At the top is a simple, unified and powerful, public facing Bus APIs (known as rbus APIs), meant to be used by different apps.
- Rbus daemon is the rtMessage routing daemon.

Features of R-Bus

- Simplified APIs
- Pub / Sub (event / notification) support
- Methods support
- Aligns well with TR 369 (USP)

* **Note**: R-Bus implementation is in progress.