determined by the application. This specification does not define a communication byte order because MCAPI is an API specification. Since MCAPI does not currently define a wire protocol (there may be a future Multicore Association specification which does specify that) then endian-ness and alignment issues etc are not relevant to this specification.

Connectionless communication allows an endpoint to send or receive messages with one or more endpoints elsewhere in the communication topology. A given message can be sent to a single endpoint (unicast). Multicast and broadcast messaging modes are not supported by MCAPI, but can be built on top of it.

Connection-oriented communication allows an endpoint to establish a socket-like streaming connection to a peer endpoint elsewhere in the communication topology, and then send data to that peer. A connection is established using an explicit handshake mechanism prior to sending or receiving any application data. Once a connection has been established, it remains active for highly efficient data transfers until it is terminated by one of the sides, or until the communication path between the endpoints is severed (for example, by the failure of the node or link which one of the endpoints is utilizing).

Connection-oriented communication over MCAPI channels is designed to be reliable, in that an application can send data over a connection and assume that the data will be delivered to the specified destination as long as that destination is reachable.

THINGS TO REMEMBER:
- In an MCAPI communication topology where different nodes may be running on different CPU types and/or operating systems, applications must ensure that the internal structure of a message is well-defined, and account for any differences in message content endian-ness, field size and field alignment.

3.4.2 MCAPI Communication Topology (ABB Extension)

The MCAPI specification provides the programmatic building blocks to configure the topology but this approach quickly becomes confusing and difficult to manage. As an alternative a declarative representation was adopted. The text box in Figure 1 below shows a partial example where endpoints compose into channels that compose into links. This terminology draws from the telecommunications domain where:

- Link – point-to-point, broadcast, multipoint, point to multipoint communications,
- Channel – used to convey an information signal from sender to receiver, and
- Endpoint – interface exposed by a communication channel.

```
<topology>
<endpoint domain="1" node="2" port="1" name="Client1_Server_MsgSEP"/>
<endpoint domain="1" node="3" port="2" name="Client2_Server_MsgSEP"/>
<endpoint domain="1" node="2" port="3" name="Client1_Server_MsgLEP"/>
<endpoint domain="1" node="3" port="4" name="Client1_Server_MsgLEP"/>
<endpoint domain="1" node="4" port="5" name="Server_Client1_MsgLEP"/>
<endpoint domain="1" node="4" port="6" name="Server_Client2_MsgLEP"/>
<endpoint domain="1" node="4" port="7" name="Server_Client1_MsgSEP"/>
<endpoint domain="1" node="4" port="8" name="Server_Client2_MsgSEP"/>
…
<channel name="Client1_Server_MsgSnd" type="msg">
      <from endpoint="Client1_Server_MsgSEP"/>
      <to endpoint="Server_Client1_MsgLEP"/>
</channel>
…
<link name="Client1_Msg" type="fullduplex">
      <send channel="Client1_Server_MsgSnd"/>
      <ack channel="Server_Client1_MsgAck"/>
</link>
…
</topology>
```

Figure 1. Example Message Topology

The message topology is static for the life cycle of the device configuration. Node assignment to a task or process is independent of topology, and can vary based on the technology stack and deployment configuration. Each task parses the XML message topology specification and interprets the contents based on its domain and node assignment.

The runtime can be additionally enhanced to support a task querying the configuration to discover the declared configuration and link types. Full duplex link pattern is only one of many possible, for example *P/1C, 1P/*C, *P/*C, etc. where P represents producer and C consumer. Based on the link types there can be standard operations, for example subscribe to a server task or broadcast to a set of listeners.

3.5 Data Delivery

On the surface, whether the communications use messages, packet channels or scalar channels, the data delivery in MCAPI is a simple series of steps: a sender creates and sends the data (either a buffer for messages and packet channels, or a scalar value for scalar channels), the MCAPI implementation carries the data to the specified destination, and the receiver receives and then consumes the data. In practice, this is exactly what happens most of the time. However, there are a number of places along the way where things can get complicated, and in these cases it is important for application designers to understand exactly what MCAPI will do.

The sections that follow describe the various steps performed by MCAPI during the transfer of a data packet.

3.5.1 Data Sending

The first step in sending data is to create it. The user application then sends the data using one of several mechanisms. The application supplies the data in a user buffer for messages and packet channels. The data is supplied directly as a scalar variable for scalar channels.

The most common reason MCAPI is unable to send a piece of data is because the sender passes in one or more invalid arguments to the send routine. The term "invalid" refers both to values that are never acceptable under any circumstances (such as specifying a data buffer size exceeding the maximum size allowed for a specific implementation) and to values that are not acceptable for the current sender (such as an invalid endpoint handle, or an unconnected channel).

In all of these cases the send operation will return a failure code indicating that the intended data was not sent. If the data is sent successfully, the send operation returns a success indication. Success means that the entire buffer has been sent.

THINGS TO REMEMBER: