The connectionless message and packet channel API functions have both blocking and non-blocking variants. The non-blocking variants of these MCAPI functions have "_i" appended to the function name to indicate that the function will return *immediately* and will complete in a non-blocking manner.

The non-blocking versions fill in a `mcapi_request_t` object and return control to the user before the communication operation is completed. The user can then use the `mcapi_test()`, `mcapi_wait()`, and `mcapi_wait_any()` functions to query the status of the non-blocking operation. These functions are non-destructive, meaning that no message, packet or scalar is removed from the endpoint queue by the functions. The `mcapi_test()` function is non-blocking whereas the `mcapi_wait()` and `mcapi_wait_any()` functions will block until the requested operation completes or a timeout occurs. Multiple threads should not be waiting the same request and attempting to do so will result in an error.

If a buffer of data is passed to a non-blocking operation (for example, to `mcapi_msg_send_i()` `mcapi_msg_recv_i()`, or to `mcapi_pktchan_send_i()`), that buffer should not be accessed by the user application for the duration of the non-blocking operation. That is, once a buffer has been passed to a non-blocking operation, the program may not read or write the buffer until `mcapi_test()`, `mcapi_wait()`, or `mcapi_wait_any()` have indicated completion, or until `mcapi_cancel()` has canceled the operation.

The MCAPI scalar channels API provides only blocking send and receive methods. Scalar channels are intended to provide a very low overhead interface for moving a stream of values. Non-blocking operations add overhead. The sort of streaming algorithms that take advantage of scalar channels should not require a non-blocking send or receive method; each process should simply receive a value to work on, do its work, send the result out on a channel, and repeat. Applications that require non-blocking semantics should use packet channels instead of scalar channels.

3.5.5 Message Communication (ABB Extension)

Endpoints may be configured with attributes. One of the endpoint attributes is `MCAPI_ENDP_ATTR_BUFFER_TYPE`. The default policy is FIFO as detailed above. Another buffer type is State Message. Communication between tasks can be organized in two separate classes[1]:

- State Message – only interested in processing the most up-to-date data from producers, i.e. messages can be lost, and
- Event Message or FIFO – every message must be consumed in order, i.e. no messages can be lost.

An example of a state message is two threads sharing the address to a global variable. For state messages it is possible that the producer is never blocked, overwriting previous messages whether or not they have been read. An example of an event message is a FIFO queue where ownership of an element is transferred from producer to consumer without the need to copy. Producer attempts to enter a message into the queue fail if all message buffers are in use by either producer or consumer, and consumer attempts to get the next message fail if the queue is empty or all active message buffers are owned by the producer.

The fundamental building block for communication is the one-way channel, either state or event message-based. From this foundation more complex hierarchies can be constructed, for example many-to-one fan-in or one-to-many fan-out. Order of event messages can be FIFO or priority-based allowing out-of-band communications to supersede normal processing.

3.6 MCAPI Messages

MCAPI messages provide a flexible method to transmit data between endpoints without first establishing a connection. The buffers on both sender and receiver sides must be provided by the user application. MCAPI messages may be sent with different priorities, on a per message basis. It is not allowed to send a message to a connected endpoint. Implementations may chose to prevent messages from being sent to connected endpoint or to leave it up to the application to manage this. Functionality for this may be added in a future version of MCAPI in it is therefore recommended that implementations preventing messages from being sent to connected endpoint use

[1] Kim, et.al., Efficient Adaptations of the Non-Blocking Buffer for Event Communication, Proceedings of ISORC, pp. 29-40 (2007).