CELLULAR SYSTEMS WHITE PAPER

OpenMAX IL Dshow Filter Integration

Document Revision: 0.1 DRAFT

Issue Date: 20 April 2006

Table of Contents

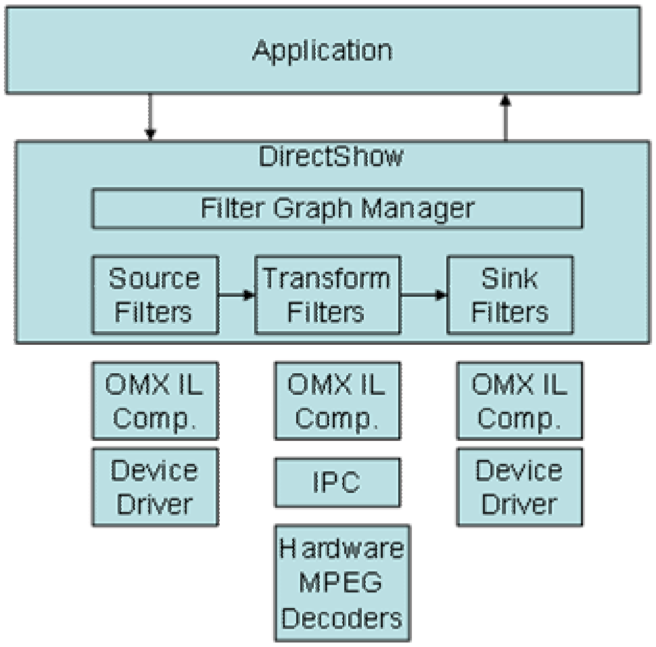
Please read the "Important Notice" on the next page.

1 Products		**2 Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

1 Introduction & Architecture

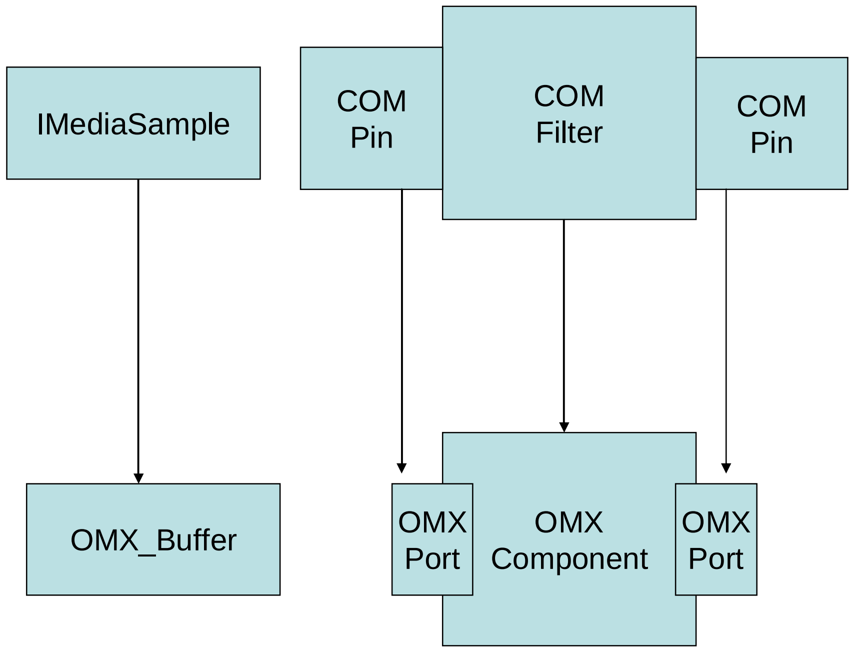
This whitepaper briefly describes how the Microsoft DirectShow Filter interface may be translated to the OpenMAX IL base profile interface for integration with Microsoft Windows Mobile Enabled devices with a pull and a push communication model.



As the diagram above depicts, a multimedia application is abstracted from the underlying hardware via the DirectShow Filter Graph which is constructed by the Filter Graph Manager. Filters in this system abstract I/O device drivers and transforms(such as codecs). These Filters are connected using Pins. It is the underlying Filter interface that this paper will map to OpenMAX IL component interface so the capture devices, decoders, and rendering interfaces are abstracted in an OS independent way.

Whereas a Filter may be in any one of three states (stopped, running, paused) an OMX base profile component can be in any one of five states (loaded, idle, executing, paused, invalid). The mapping difference between loaded and idle for OMX components is in whether or not hardware resources have been allocated. The invalid state is intended primarily for multi-core system error handling and should not be entered throughout the normal operation of an OMX component.

The architectural mapping between DirectShow filters and OpenMAX components appears in the figure below. Filter Pins correspond to OMX component ports. Although OMX ports generally share the same states as OMX components, they can be independently enabled and disabled.


IMediaSample
COM Pin
COM Filter
COM Pin
OMX_Buffer
OMX Port
OMX Component
OMX Port

2 DShow to OMX interface mapping

This section briefly outlines a workable mapping between Dshow Interfaces and OpenMAX IL interfaces.

CBaseFilter Class to OMX Mapping

The CBaseFilter class is the base class and comprises the following methods and variables:

Protected Member Variables	**OMX IL Method or Variable**	**Description**
m_State	OMX_STATETYPE	Current state of the filter.
m_pClock	Corresponds to OMX Clock Component	Pointer to the filter's reference clock.
m_tStart	NA	Reference time that corresponds to stream time 0.
m_clsid	NA	Class identifier (CLSID) of the filter.
m_pLock	NA	Pointer to a critical section that is used to serialize state changes.
m_pName	***pName***	Filter name.
m_pGraph	NA	Pointer to the filter graph manager.
m_pSink	NA	Pointer to the **IMediaEventSink** interface on the filter graph manager.
m_PinVersion	`nVersion`	Current version of the set of pins on this filter.
Public Methods		
CBaseFilter	`OMX_GetHandle`	Constructor method.
~CBaseFilter	`OMX_FreeHandle`	Destructor method.
StreamTime	Applicable only to OMX Clock Component	Retrieves the current stream time. *Virtual.*

IsActive	`OMX_GetState`	Determines whether the filter is currently active (running or paused).
IsStopped	`OMX_GetState`	Determines whether the filter is currently stopped.
NotifyEvent	`SetCallbacks(hHandle, pCallBacks, pAppData)` Sets up events from the component which may be translated into NotifyEvents.	Sends an event notification to the filter graph manager.
GetFilterGraph	NA	Retrieves a pointer to the filter graph manager.
ReconnectPin	OMX_SetupTunnel	Breaks an existing pin connection and reconnects it to the same pin, using a specified media type.
GetPinVersion	OMX_GetComponentVersion	Retrieves a version number for the set of pins on this filter. *Virtual.*
IncrementPinVersion	NA	Increments the version number on the set of pins.
GetSetupData	NA	Retrieves the registration data for the filter. *Virtual.*
Pure Virtual Methods		
GetPinCount	`OMX_GetParameter`	Retrieves the number of pins.
GetPin	`OMX_GetParameter`	Retrieves a pin.
IPersist Methods		
GetClassID	NA	Retrieves the class identifier.
IMediaFilter Methods		
GetState	`OMX_GetState`	Retrieves the filter's state (running, stopped, or paused).

SetSyncSource	NA	Sets a reference clock for the filter.
GetSyncSource	NA	Retrieves the reference clock that the filter is using.
Stop	`OMX_SendCommand(hComp, OMX_CommandStateSet, OMX_StateIdle, 0)OMX_SendCommand(hComp, OMX_CommandStateSet, OMX_StateLoaded, 0)`	First transitions the OMX component to Idle and then transitions the component to Loaded. (the OMX component does not have resources in this state)
Pause	`OMX_SendCommand(hComp, OMX_CommandStateSet, OMX_StateIdle, 0) (when stopped)` `OMX_SendCommand(hComp, OMX_CommandStateSet, OMX_StatePaused, 0)`	Pauses the filter if it is executing. If the filter is stopped, the OMX component is first transitioned to Idle and then to Paused. (the OMX component has resources in this state – it is always required to Pause before Running.)
Run	`OMX_SendCommand(hComp, OMX_CommandStateSet, OMX_StateExecuting, 0)`	Runs the filter.
IBaseFilter Methods		
EnumPins	NA	Enumerates the pins on this filter.
FindPin	NA	Retrieves the pin with the specified identifier.
QueryFilterInfo	`OMX_GetParameter`	Retrieves information about the filter.
JoinFilterGraph	NA	Notifies the filter that it has joined or left a filter graph.
QueryVendorInfo	NA	Retrieves a string containing vendor information.
IAMovieSetup Merthods		
Register	NA	Adds the filter to the registry.
Unregister	NA	Removes the filter from the

		registry.

CBaseInputPin to OMX Port Mapping

Protected Member Variables	**OMX IL Method or Variable**	**Description**
m_pAllocator	Points to OMX_AllocateBuffer	Pointer to the memory allocator.
m_bReadOnly	NA	Flag that indicates whether the allocator produces read-only media samples.
m_bFlushing	NA	Flag that indicates whether the pin is currently flushing.
m_SampleProps	NA	Properties of the most recent sample.
Public Methods		
CBaseInputPin	NA	Constructor method.
~CBaseInputPin	NA	Destructor method.
BreakConnect	`OMX_CommandPortDisable`	Releases the pin from a connection.
IsReadOnly	NA	Queries whether the allocator uses read-only media samples.
IsFlushing	NA	Queries whether the filter is currently flushing.
CheckStreaming	NA	Determines whether the pin can accept samples. *Virtual.*
PassNotify	NA	Passes a quality-control message to the appropriate object.
Inactive	NA	Notifies the pin that the filter is no longer active. *Virtual.*

SampleProps	NA	Retrieves the properties of the most recent sample.
IPin Methods		
BeginFlush	OMX_CommandFlush	Begins a flush operation.
EndFlush	NA	Ends a flush operation.
IMemInputPin Methods	(Used in Push Communication)	
GetAllocator	OMX_UseBuffer	Retrieves the memory allocator proposed by this pin.
NotifyAllocator	NA	Specifies an allocator for the connection.
GetAllocatorRequirements	OMX_GetParameter	Retrieves the allocator properties requested by the input pin.
Receive	OMX_EmptyThisBuffer	Receives the next media sample in the stream.
ReceiveMultiple	NA	Receives multiple samples in the stream.
ReceiveCanBlock	NA	Determines whether calls to the **CBaseInputPin::Receive** method might block.
IQualityControl Methods		
Notify	NA	Receives a quality-control message.

CBaseOutputPin to OMX Port Mapping

Protected Member Variables	**OMX IL Method or Variable**	**Description**
m_pAllocator	Points to OMX_AllocateBuffers	Pointer to the memory allocator.

m_pInputPin	NA	Pointer to the input pin connected to this pin.
Public Methods		
CBaseOutputPin	NA	Constructor method.
CompleteConnect	`ComponentTunnelRequest`	Completes a connection to an input pin. *Virtual.*
DecideAllocator	Determines whether OMX_UseBuffers or OMX_AllocateBuffers is called	Selects a memory allocator. *Virtual.*
GetDeliveryBuffer	(push communication) OMX_FillThisBuffer	Retrieves a media sample that contains an empty buffer. *Virtual.*
Deliver	(push communication) OMX_FillBufferDone	Delivers a media sample to the connected input pin. *Virtual.*
InitAllocator	NA	Creates a memory allocator. *Virtual.*
CheckConnect	ComponentTunnelRequest	Determines whether a pin connection is suitable.
BreakConnect	OMX_PortDisable	Releases the pin from a connection.
Active	NA	Notifies the pin that the filter is now active.
Inactive	NA	Notifies the pin that the filter is no longer active.
DeliverEndOfStream	NA	Delivers an end-of-stream notification to the connected input pin. *Virtual.*
DeliverBeginFlush	NA	Requests the connected input pin to begin a flush operation. *Virtual.*
DeliverEndFlush	NA	Requests the connected input pin

		to end a flush operation. *Virtual.*
DeliverNewSegment	NA	Delivers a new-segment notification to the connected input pin. *Virtual.*
Pure Virtual Methods		
DecideBufferSize	NA	Sets the buffer requirements.
IPin Methods		
BeginFlush	OMX_CommandFlush	Begins a flush operation.
EndFlush	NA	Ends a flush operation.
EndOfStream	NA	Notifies the pin that no additional data is expected.

IMemAllocator to OMX IL Mapping

Method	**OMX IL Method**	**Description**
SetProperties	OMX_SetParameter	Specifies the number of buffers to allocate and the size of each buffer.
GetProperties	OMX_GetParameter	Retrieves the number of buffers that the allocator will create, and the buffer properties.
Commit	OMX_CommandPortEnable	Allocates the buffer memory.
Decommit	OMX_CommandPortDisable	Releases the buffer memory.
GetBuffer	This method will block until the underlying OMX component port has received and processed a buffer when no other buffers are available.	Retrieves a media sample that contains an empty buffer.
ReleaseBuffer	This method releases a buffer to the allocation pool.	Releases a media sample.

IAsyncReader to OMX IL Mapping (Used in Pull Communication)

Method	**OMX IL Method**	**Description**
BeginFlush	OMX_CommandFlush	Causes all outstanding reads to return.
EndFlush	NA	Ends the flushing operation.
Length	OMX_GetParameter	Retrieves the total length of the stream, and the currently available length.
RequestAllocator	Returns pointer to allocator which calls an underlying OMX_AllocateBuffers	Retrieves the actual allocator to be used.
Request	OMX_FillThisBuffer	Queues a request for data.
SyncReadAligned	NA	Performs an aligned synchronized read.
SyncRead	NA	Performs a synchronized read.
WaitForNext	OMX_FillBufferDone unblocks this call.	Blocks until the next sample is completed or the time-out occurs.

3 Buffer Management & Communication

The buffer management scenario between two OMX based filters varies dependant on whether a push or pull model is desired. This paper assumes a pull model is adopted. Although all OMX components must be capable of both allocating or using externally allocated buffers, a specific allocation method may be desired in order to maximize memory efficiency or performance. The allocator between output and input pin must be negotiated.

One of the primary differences between DirectShow filters and OMX components is that filters are implemented as COM objects with pins that are also implemented as COM objects. OMX components have “ports” which although considered part of the component, have two independent states (enabled(resources allocated), disabled(resources de-allocated)) and also have independent buffering and configuration interfaces within the component.

For pull communication, the IAsyncReader interface is used as follows:

1. The input pin begins the negotiation by calling **IAsyncReader::RequestAllocator** on the output pin. The input pin provides its buffer requirements and a pointer to an allocator which calls the underlying OMX_AllocateBuffer method.

2. The output pin must then select an allocator. It has the option of using the one provided by the input pin, if any, or it can create its own. If it creates its own, it will provide a pointer to an allocator which calls the underlying OMX_AllocateBuffers method. If it chooses to use the input pin allocator, it will provide a pointer to an allocator which call the underlying OMX_UseBuffers method.

3. The output pin must return the allocator as an outgoing parameter in the **RequestAllocator** method. The input pin should check the allocator properties by making a call to OMX_GetParameter on the associated port.

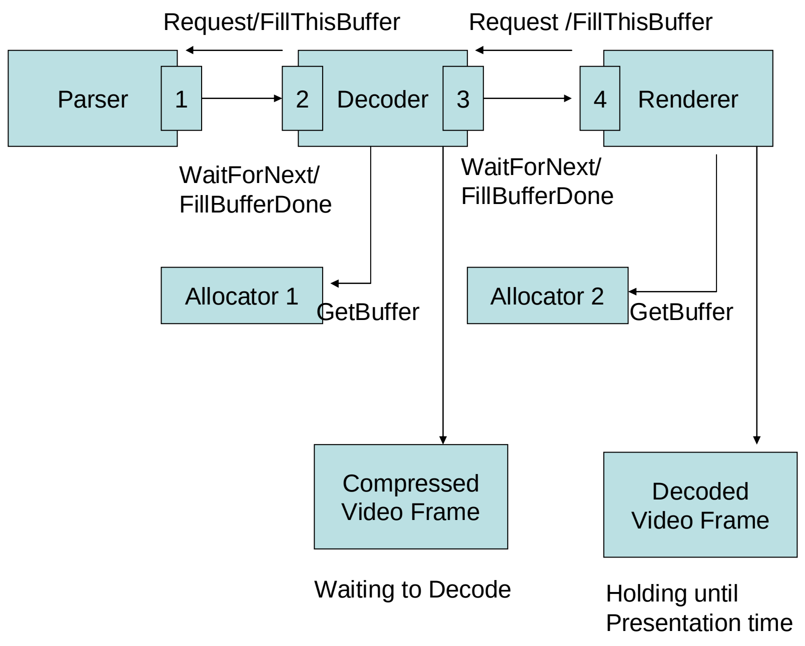
4. The input pin is responsible for allocating resources. The input pin is responsible for calling the output port's allocator (underlying OMX_AllocateBuffers or OMX_UseBuffers method) and enabling or disabling both the input and output ports.

5. At any time during the negoriation process, either pin can fail the connection requiring an alternate component to be used or an event to be generated to the Filter Graph Manager.

6. If the output pin uses the input pin's allocator, it can use that allocator only for samples delivered to that input pin. This is also true for the underlying OMX components – allocated buffers may only be exchanged between the associated input and output ports.

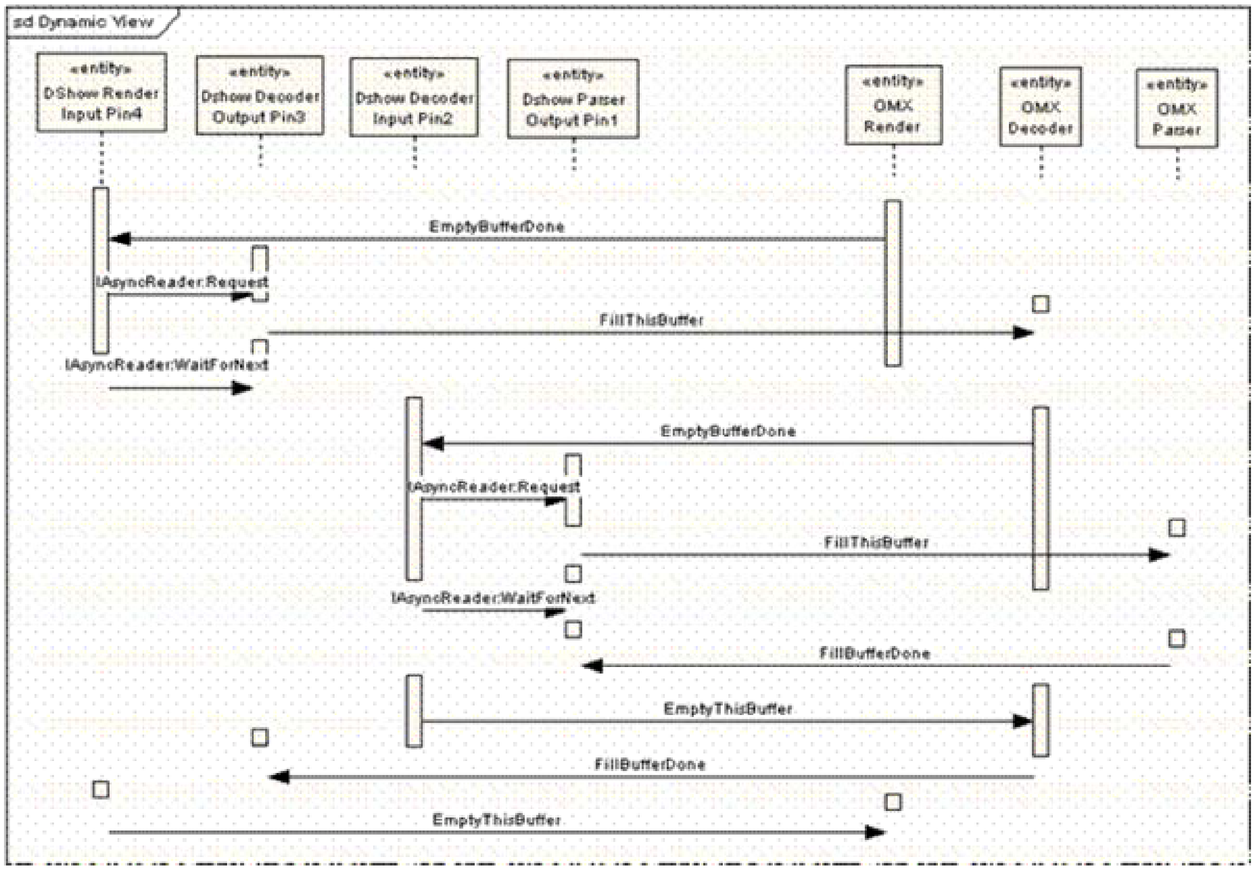
The output pin (or port) in the scenario above is responsible for determining whether to allocate buffers locally or at the input. For pull communications the input pin or port is required to allocate/de-allocate/commit/decommit buffers.

Buffer communication in a DirectShow filter graph utilizing pull communication might take place as depicted below with OMX IL integrated components. While parser configuration parameters are not specified in OMX IL 1.0, a parser component may still be implemented using the standard interfaces.



For OpenMAX pull components, FillBufferDone callbacks are made to trigger the use and subsequent re-circulation of buffers to the designated allocator (where the allocator in the above example is the input pin for all connections). Although the allocator in an underlying OpenMAX component may be part of the input or output port, we have adopted the convention where the allocator is part of the input port for this example. Throughout this document, a Pin refers to the Microsoft specified part of a DirectShow Filter while a Port refers to the Khronos specified part of an OpenMAX IL Component.

The sequence diagram below details how the above configuration might work for pull communication.

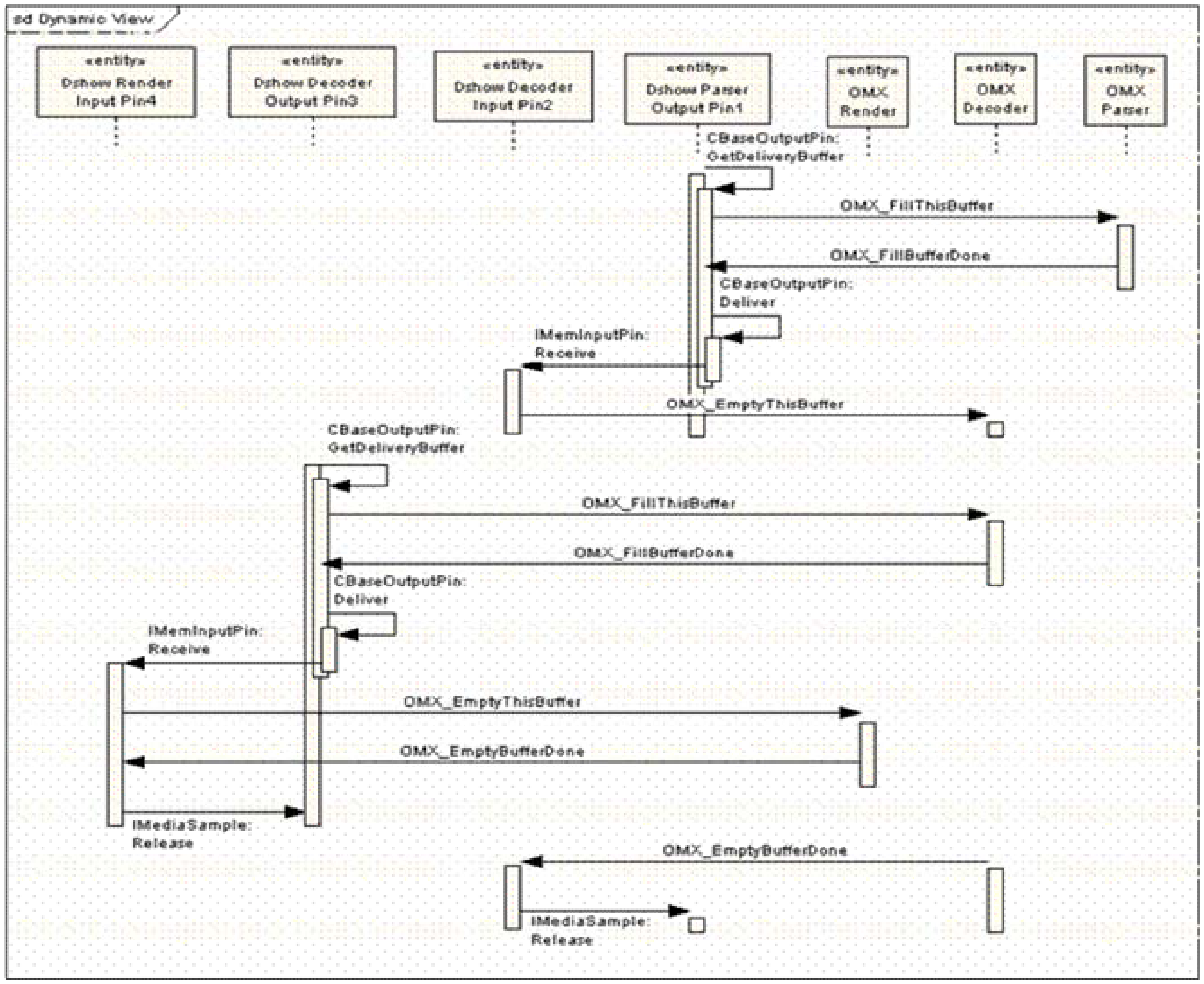


In an **IMemInputPin** connection(where push communication is desired), allocator negotiation works as follows:

1. Optionally, the output pin calls **IMemInputPin::GetAllocatorRequirements**. This method retrieves the input pin's buffer requirements, such as memory alignment. The input pin calls OMX_GetParameter to retrieve its port buffer requirements.
2. Optionally, the output pin calls **IMemInputPin::GetAllocator**. This method requests an allocator from the input pin. The input pin provides one, or returns an error code. If one is provided, the underlying OMX_AllocateBuffers method may be called during allocation.
3. The output pin selects an allocator. It can use one provided by the input pin, or create its own. In the following example we assume the output pin uses its own allocator by calling its underlying OMX_AllocateBuffers method.

4. The output pin calls **IMemAllocator::SetProperties** to set the allocator properties. However, the allocator might not honor the requested properties. (For example, this can happen if the input pin provides the allocator.) The allocator returns the actual properties as an output parameter in the **SetProperties** method. This translates into an OMX_SetParam call.
5. The outpin calls **IMemInputPin::NotifyAllocator** to inform the input pin of the selection.
6. The input pin should call **IMemAllocator::GetProperties** to verify whether the allocator properties are acceptable. This translates into an OMX_GetParameter call.
7. The output pin is responsible for committing and decommitting the allocator. This occurs when streaming starts and stops. This translates into enabling or disabling a given port.

In the sequence diagram below, it is assumed that the parser's output port is the allocator. This diagram represents the same filter graph constructed for the pull example but is shown below utilizing the IMemInputPin(instead of IAsyncReader) interfaces with push communication.



4 Building Filter Graphs

DirectShow filter graphs are built using IFilterGraph and the inherited class IGraphBuilder. Although these classes don't map to OpenMAX components, some of the methods in these classes can leverage OpenMAX Core Methods when OMX Filters are in use. The IGraphBuilder:Connect method or IFilterGraph:ConnectDirect method may make an underlying OMX_SetupTunnel call.