iCC 1995

2nd international CAN Conference

in London (United Kingdom)

Sponsored by

Motorola Semiconductor
National Semiconductor
Philips Semiconductors

Organized by

CAN in Automation (CiA)
international users and manufacturers group
Am Weichselgarten 26
D-91058 Erlangen
Phone +49-9131-69086-0
Fax +49-9131-69086-79
Email:headquarters@can-cia.de
URL: http://www.can-cia.de

DeviceNet Transports and Data Production Triggers:

Authors: Jeff Hanneman, Pat Maloney, Dan Noonen, Stuart Siegel

Allen-Bradley 1 Allen-Bradley Dr. Mayfield Heights, Ohio 44124

(216) 646-5000

Abstract:

DeviceNet connections can be configured with a multitude of behaviors based upon the *transport class types* and *data production triggers* desired by the end communicating applications. The DeviceNet communication model provides a variety of connection object transport class types and data production triggers. Connection objects can be configured to produce only or consume only. They can be configured to both produce and consume and to do so in a fashion that is either synchronous or asynchronous with the end application to which they are linked. DeviceNet connections are capable of producing their data based upon a connection timer (cyclically), upon a change-of-state of the I/O or at the application's discretion. With this flexibility an end device may set up a myriad of communication links that are useful in architectures including Master/Slave, Peer-to-Peer and Distributed Control. This paper will present the communications links that can be established using DeviceNet connection objects and give examples as to how these communications links can be applied in the various network architectures.

Introduction:

DeviceNet models individual devices as a collection of objects. An object is an abstract representation of a particular functionality or behavior within a device. Some of a device's objects embody that device's application functionality (**Application Objects**) while others are responsible for handling communications into and out of the device (**Connection Objects**). This paper will focus on Connection Objects and how they are used to transport data in a variety of network architectures.

The DeviceNet protocol employs a ***Client/Server*** connection based scheme to facilitate the movement of data between end applications. A *Client Connection object* and one or more *Server Connection Object(s)* are associated with every DeviceNet connection. A Client Connection Object originates data productions and a Server Connection Object consumes and reacts to that data. The Connection Objects associated with a DeviceNet connection are flexible and can be configured with a variety of behaviors. Two important aspects of a Connection Object's behavior are:

- **The connection's ability to produce and/or consume data in a fashion that is either synchronous or asynchronous with the end application -and-**
- **The connection's ability to process a variety of events that trigger the production of data**

The first Connection Object behavioral aspect is governed by the connection's ***Transport Class*** attribute. The Transport Class attribute indicates whether the connection will produce data, consume data or both. In addition, if a connection is to consume first and then produce (server device), this attribute will indicate whether or not the application object will be required to process the consumed data before triggering a data production. The second Connection Object behavioral aspect is governed by the connection's ***Production Trigger*** attribute. This attribute indicates how a client device connection initiates data

production. A client device may produce its data based on a *cyclic timer*, a *change of state* of the data or an *application object signal*.

By manipulating these two attributes a connection can be created that exhibits the behavior required by the device's application and the system architecture in which the device must operate. Connections can be created that provide for the movement of data in a point-to-point fashion (from 1 client to 1 server) or in a multicast fashion (from 1 client to 2 or more servers). Also, the data productions of a client device can be either acknowledged or unacknowledged. With this flexibility, connections can be created that allow devices to operate in a Master/Slave, a Peer-to-Peer or a Highly Distributed architecture.

Connection Types:

There are four basic types of connections that can be established by configuring DeviceNet Connection Objects. These connections are:

1. Point-to-Point without acknowledgment
2. Point-to-Point with acknowledgment (either server application synchronous or server application asynchronous)
3. Multicast without acknowledgment
4. Multicast with acknowledgment (either server application synchronous or server application asynchronous)

Figure 1 shows a typical Point-to-Point connection without acknowledgment. Note that the client device contains a producing only Connection Object (***Client Transport Class 0***) and the server device contains a consuming only Connection Object (***Server Transport Class 0***). The data production trigger utilized by the client device can be any one of the three mentioned above. This type of connection would be well suited for use in either a Peer-to-Peer or a Distributed architecture. The client device could represent a sensor and the server device could represent a controller or a controller/actuator pair.

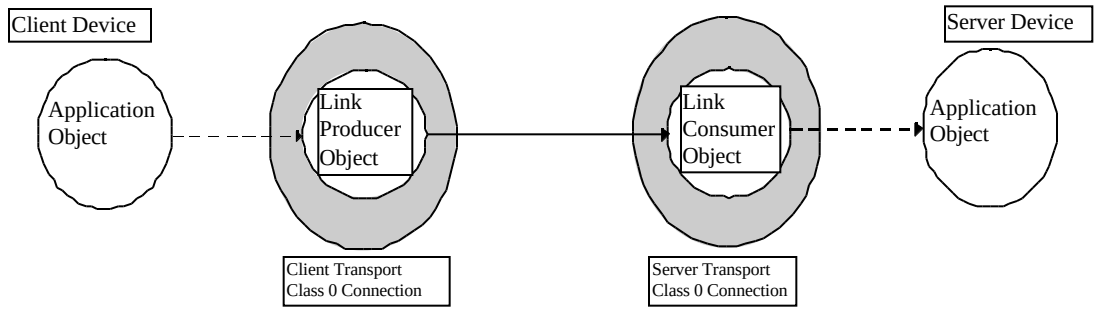

Figure 1. Point-to-Point Connection Without Acknowledgment

Figure 2 shows a typical Point-to-Point connection with acknowledgment. The client device contains a ***Client Transport Class 2 or 3*** Connection Object that both produces and consumes. It is important to note that since the client end of a connection always initiates a data production, there is no distinction between a ***Client Transport Class 2*** Connection Object and a ***Client Transport Class 3*** Connection Object. In other words, the client application object(s) will always be synchronized with the client Connection Object.

It should also be noted that the server device in figure 2 contains a ***Server Transport Class 2*** Connection Object that both consumes and produces and it does so in a fashion that is asynchronous of the Application Object. This is represented by the dashed line that connects the Link Consumer Object directly to the Link Producer Object. As a *Server Transport Class 2* Connection Object, the act of consuming immediately triggers a response. The response may optionally contain data, however, the data is provided by the Application Object asynchronous of the Connection Object's operation. If, as in figure

3, the server device Connection Object had been configured as a ***Server Transport Class 3*** Connection Object, the Application Object would have received the consumed data before triggering the Connection Object to produce a response (synchronous operation). Synchronous operation provides the Application Object with the opportunity to validate and/or process the consumed data before triggering a data production. These two server transport types differ in the types of acknowledgment that each transport can generate. A *Server Transport Class 2* Connection Object produces acknowledgments generated by the Connection Object (*OSI Model: Transport Layer acknowledgment*) while a *Server Transport 3* Connection Object produces acknowledgments generated by the Application Object (*OSI Model: Application Layer acknowledgment*).

Again the data production trigger utilized by the client device can be the expiration of a cyclic timer, a change of state of the data, or an application object specific trigger. This type of connection would be well suited for use in any of the previously mentioned architectures. In a Master/Slave architecture the client device would represent the master and the server device could represent a sensor or an actuator. In Peer-to-Peer or Distributed architectures the connection could be used as shown above in figure 1 with the acknowledgment added to insure data integrity.

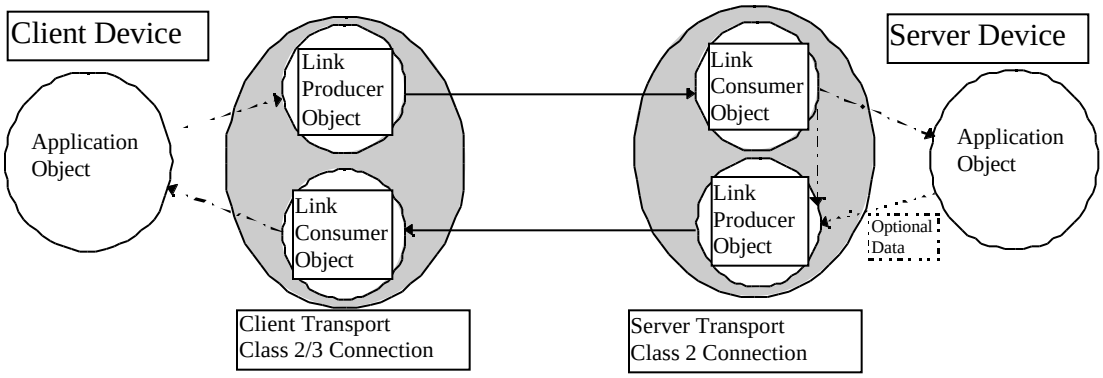

Figure 2. Acknowledged Point-to-Point Connection (Server Application Asynchronous)

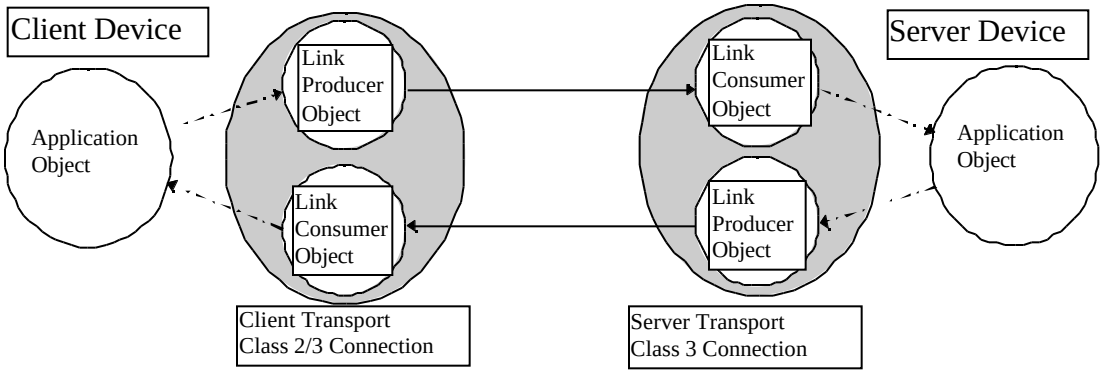

Figure 3. Acknowledged Point-to-Point Connection (Server Application Synchronous)

Figure 4 shows a Mulitcast connection without acknowledgment. The client device contains a producing only *Client Transport Class 0* Connection Object and the server devices each contain a consuming only *Server Transport Class 0* Connection Object. This is similar to the Point-to-Point scenario shown in figure 1 and, in fact, the individual Connection Objects are configured identically. Again the data production trigger utilized by the client device can be any one of the three previously mentioned triggers. This type of connection would be well suited for use in Peer-to-Peer or Distributed architectures. The client device

could be producing a sensor input or a diagnostic/alarm signal and the n server devices could be actuating outputs, executing control algorithms or responding to system alarms.

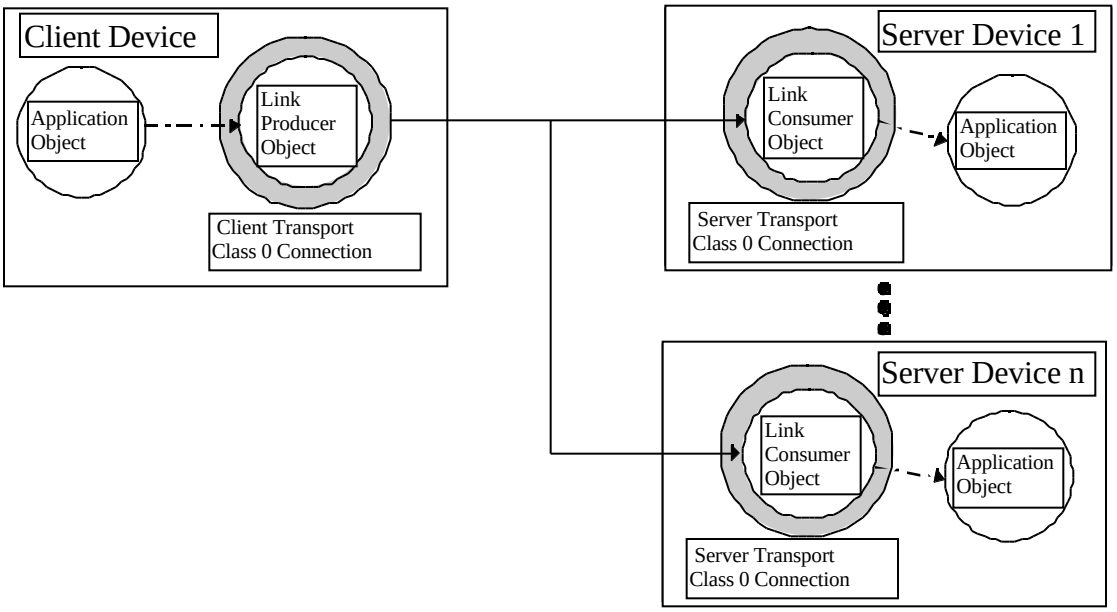

Figure 4. Unacknowledged Multicast Connection

Figure 5 shows a Multicast connection with acknowledgment. The client device contains a single *Client Transport Class 0* Connection Object responsible for data production and n *Server Transport Class 0* Connection Objects responsible for the consumption of acknowledgments from the n server devices. It is the Application Object, however, that is ultimately responsible for verifying the reception of all acknowledgments and, if necessary, executing error recovery logic. The n server devices each contain Connection Objects that interact with their respective Application Objects in a synchronous fashion (*Server Transport Class 3*). Once again the data production trigger utilized by the client device can be any one of the three previously mentioned production triggers. This type of connection would be well suited for use in any of the stated architectures. In a Master/Slave architecture the client device would represent the master and the server devices would represent multiple sensors or actuators (slaves). This connection type is presently defined in the DeviceNet specification for use as the master's Bit-Strobed I/O connection. In Peer-to-Peer or Distributed architectures an acknowledged multicast connection could be used by any client device requiring confirmed transmission of its data to multiple devices.

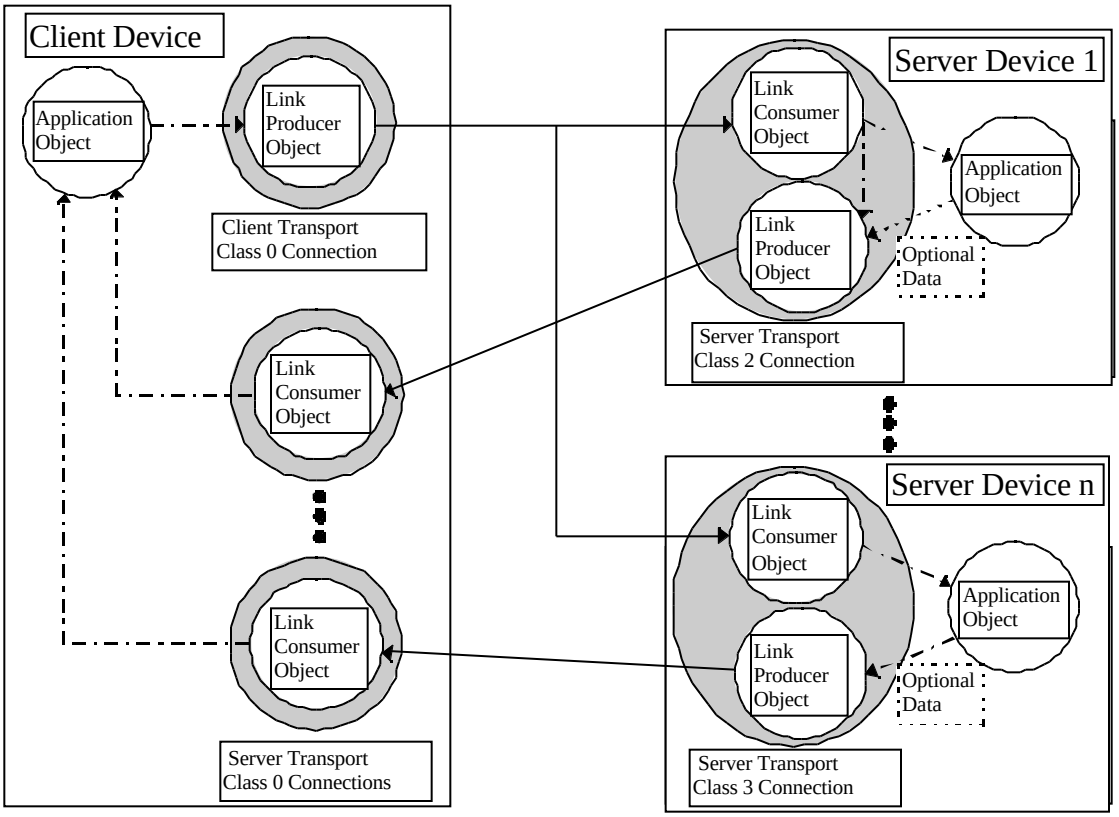

Figure 5. Acknowledged Multicast Connection

DeviceNet Connections and Control Architectures:

DeviceNet makes no assumptions about the control architectures in which it is to be applied. The protocol's connection based scheme is equally applicable regardless of whether your control architecture is Master/Slave, Peer-to-Peer or Highly Distributed. In this section examples will be presented showing the application of the DeviceNet protocol in each of these control architectures.

Master/Slave:

The Master/Slave architecture is possibly the best understood of all the control architectures. In its simplest form a central "controller" device performs data acquisition from field sensors, executes control algorithms based on those sensor inputs and then updates the output values of field actuators. An example of this is shown in figure 6. Figure 6 shows a master device performing an I/O gathering and distributing function along with the execution of a master application. All of a master's I/O operations can be supported by two connection types: Multicast with acknowledge and Point-to-Point with acknowledge. The Master/Slave portion of the DeviceNet specification refers to these two connection types as the Bit-Strobed I/O Connection and the Polled I/O Connection respectively.

Referring to figure 6 we see that the master device Application Object triggers the production of a Bit-Strobe request message across an ***acknowledged multicast connection***. That message is consumed by all devices on the network that are configured to do so (devices 1 and 2 in our example). Those devices then respond with input, status or a simple acknowledge. Note that the Bit-Strobed devices in our example have been shown as inputs. The Bit-Strobed devices could just as easily have been shown as outputs as well.

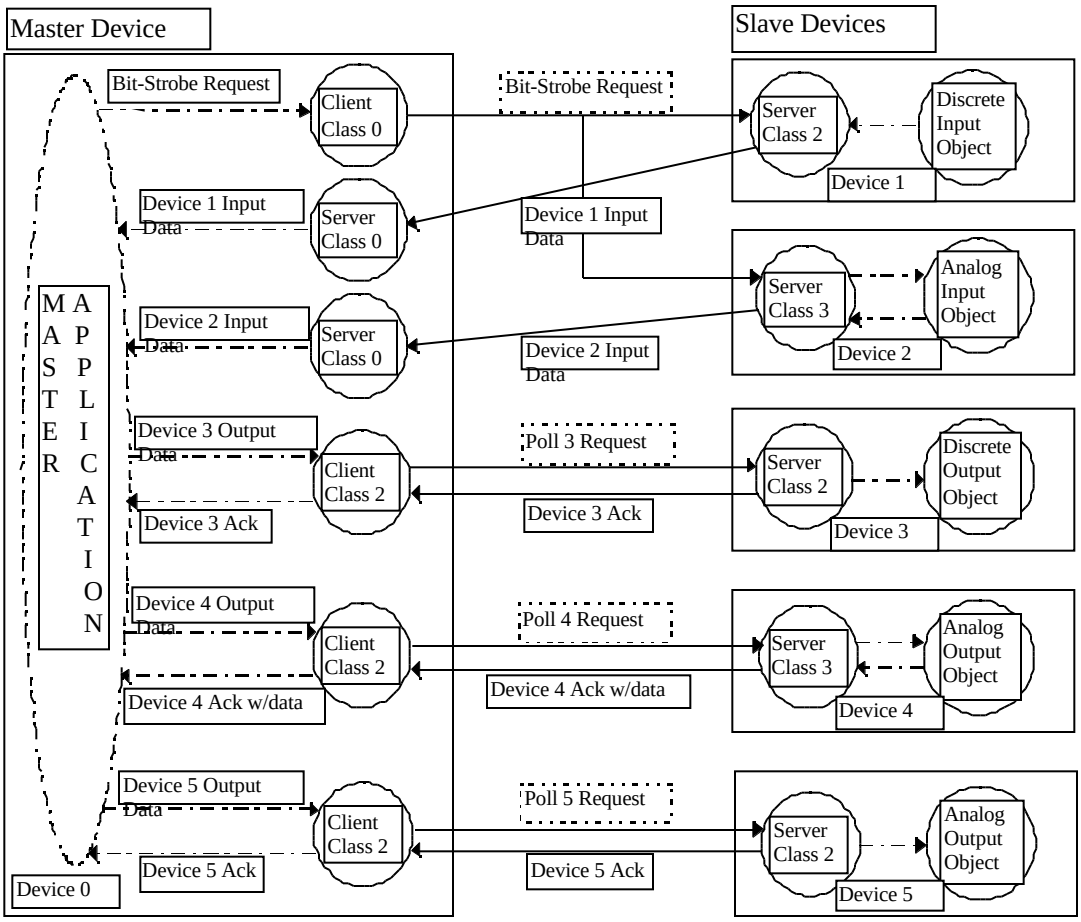

Figure 6. DeviceNet Connections in a Master/Slave Architecture

The remaining devices in the example (devices 3,4 and 5) are communicated to via ***acknowledged point-to-point*** connections. The master device Application Object triggers the production of a poll request message containing output data for each of the devices. The devices respond with an acknowledgment and optional data. The polled devices in our example have been shown as outputs but they could have been inputs as well. If they had been inputs, the poll request message could have acted as a master generated trigger causing the slaves to produce their input values.

Peer-to-Peer:

The use of a Peer-to-Peer architecture provides a greater degree of design flexibility to the control system engineer as compared to the Master/Slave architecture. For example, main control elements throughout the system can share I/O and configuration data. This allows for a clean segmentation of the control solution while, at the same time, allowing for the use of smaller, less costly controllers. Also, the traditional roles of master and slave become blurred as I/O devices gain the capability of reporting their data at their own discretion. I/O devices no longer need to be triggered to report by a master. This provides for a more efficient use of the network's bandwidth.

An example of a Peer-to-Peer network is shown in figure 7. In this example a tool device shares the same network with two small Master/Slave control schemes. The tool device provides the control system engineer with a user interface to configure the master controllers. The master devices are each responsible for gathering and distributing I/O data from/to their respective slaves and controlling their own subprocesses. This example shows the Device 1 master sharing data with the Device 3 master via an acknowledged point-to-point connection. The shared data consumed by the Device 3 master allows it to make control decisions in its own subprocess based on the state of the Device 1 master's subprocess.

This capability can be exploited in the control of batch processes. Finally, an important point to be noted in this example is that device 6 is a client device and reports input data to master device 3 based upon its own internal trigger (I/O data change-of-state, cyclic timer expiration, etc.). This capability results in fewer packets traversing the bus for a given amount of I/O when compared to a similar master/slave architecture.

It is interesting to note that from a DeviceNet communications modeling point of view many of the connections utilized in the peer-to-peer architecture are identical to those found in the master/slave architecture.

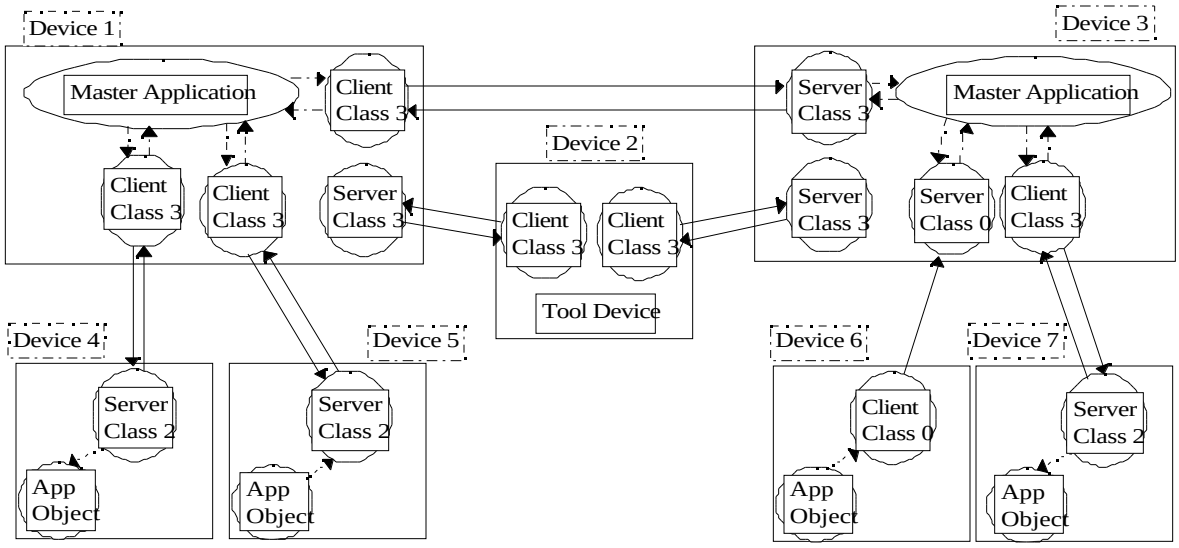

Figure 7. DeviceNet Connections in a Peer-to-Peer Architecture

Highly Distributed Control:

Highly Distributed Control has been talked about for the past couple of decades but, until recently, cost effective technology with which one could implement a highly distributed control scheme simply didn't exist. The emergence of highly capable, low cost networking technologies coupled with "smarter" and lower cost I/O products puts highly distributed architectures within reach. The distribution of I/O in a distributed control architecture provides for a more efficient use of bus bandwidth and shorter end to end delays for I/O data thus allowing for tighter closed loop control. Also, the distribution of control reduces or even eliminates the need for expensive dedicated controllers.

An example of a Highly Distributed Control architecture is shown in figure 8. This example shows the application of Highly Distributed Control to a conveyor line. In this example a presence sensing device detects the presence of a box as it moves down the conveyor line. The detection of a box triggers the presence sensor's client connection to send a "Box Present" message to the bar code scanner via an ***acknowledged point-to-point*** connection. The consumption of that message causes the bar code scanner to scan the box which, in turn, triggers the bar code scanner's client connection to trigger the production of a "Box_Type(A|B) Detected" message. That message is communicated to the linear actuator via another ***acknowledged point-to-point*** connection. Finally, the linear actuator routes the box to one of two successive conveyor lines based upon the consumed box type data. Note that on both of the acknowledged point-to-point connections the client Connection Objects send an indication back to their respective Application Objects indicating that the acknowledge had been received. If an acknowledge had not been received within some application specific time, the client Connection Objects could have been triggered to produce again.

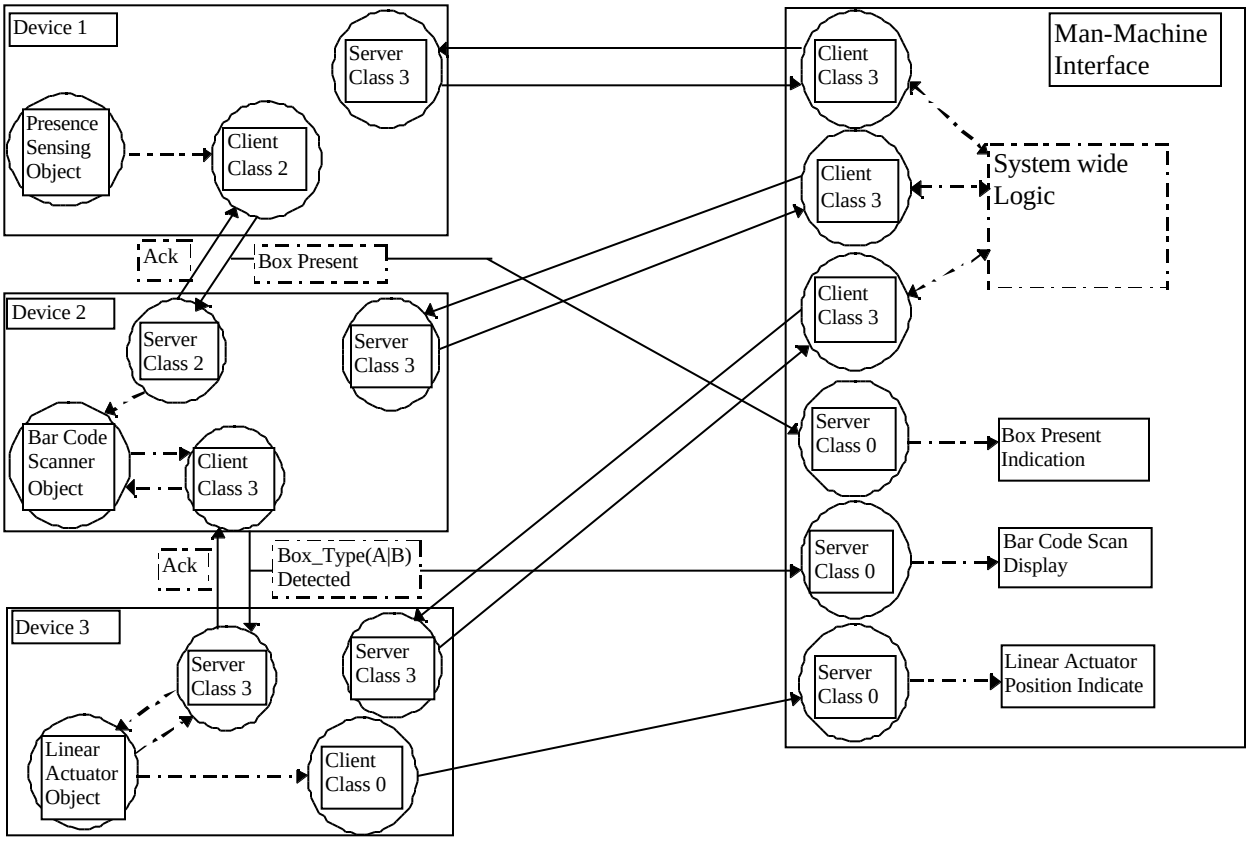

Figure 8. DeviceNet Connections in a Distributed Architecture (Configuration Time)

A Man-Machine Interface (MMI) is added to the system to provide the control system engineer with a "window" into the distributed process. System wide control algorithms are configured/entered by the control engineer at the MMI and then the MMI is responsible for "breaking down" the control algorithms and distributing the algorithm fragments to the appropriate devices throughout the system. In our simple conveyor line example the MMI is responsible for creating and configuring the necessary Connection Objects to route the I/O signals throughout the system. The MMI is also responsible for configuring the distributed control elements (i.e. Presence Sensing Object, Bar Code Scanner Object and Linear Actuator Object).

The acknowledged point-to-point connections between the I/O devices are used to route runtime I/O data from device to device. Notice, however, the *Client Class 3* Connection Objects within the MMI and S*erver Class 3* Connection Objects within each of the devices. These Connection Objects are the client and server end points respectively of acknowledged point-to-point connections between the MMI and the I/O devices. These connections provide the MMI with a configuration channel into each of the I/O devices. The MMI will use these connections to create and configure device Connection Objects used to transfer runtime I/O data as well as configure device Application Object attributes. These configuration channel connections are needed only at configuration time and may be deleted after use. The DeviceNet Connection Object Class supports service primitives that provide for the dynamic creation and deletion of individual Connection Objects. Thus, Connection Objects (and valuable device resources) need only be in use when they are needed. Figure 9 shows this same conveyor line control example after all system configuration functions have been performed.

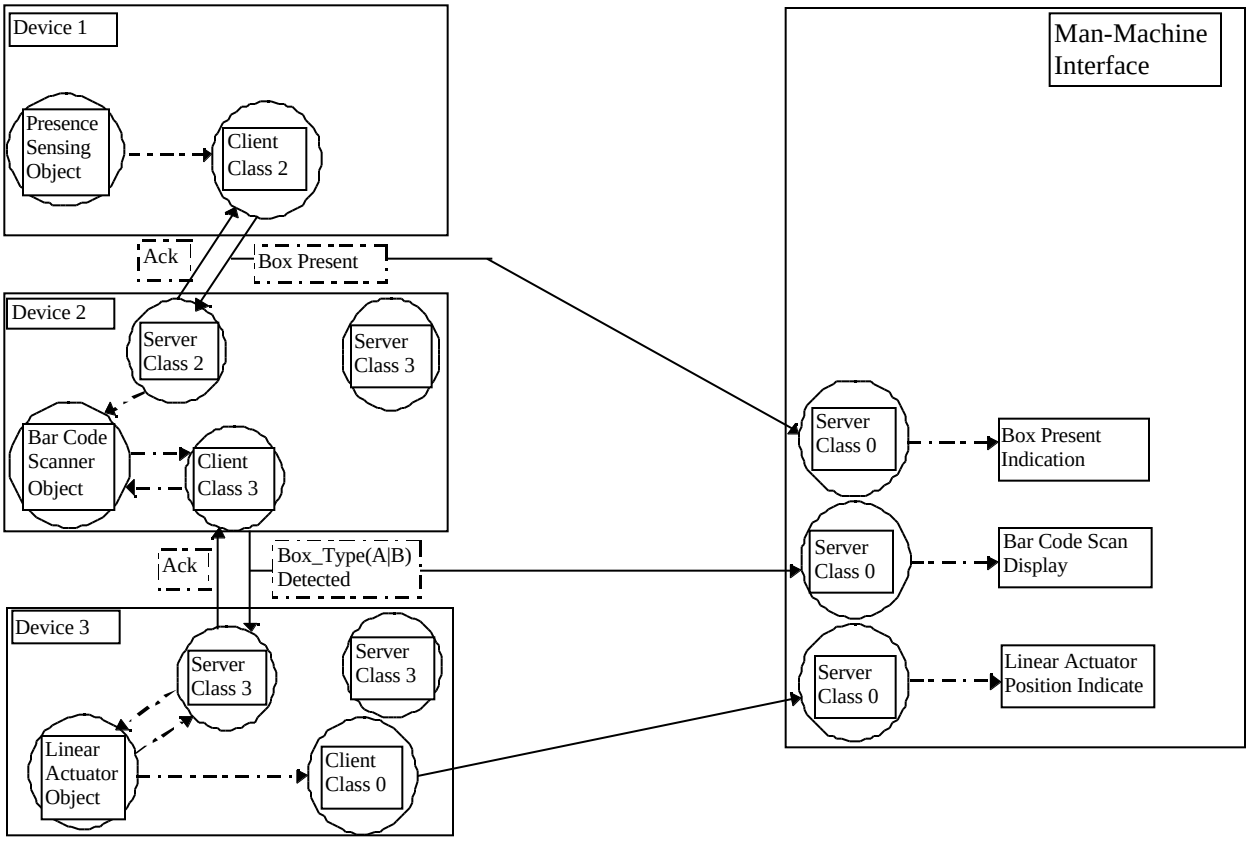

Figure 9. DeviceNet Connections in a Distributed Architecture (Run Time)

Finally, take note of the *Server Class 0* Connection Objects contained within the MMI. These Connection Objects serve to monitor the I/O traffic on the bus and route the individual I/O data points to display panels or some other supervisory application object. These Connection Objects are used to monitor bus traffic only and thus consume no bus bandwidth!

Here again it should be noted that from a communications modeling point of view the connections utilized in the highly distributed architecture are identical to the connections utilized in both the peer-to-peer and master/slave architectures.

Hybrid Networks:

By examining each of the previous architecture examples we see that the communication model is virtually identical for all architectures. In fact, the connection objects contained within the end devices are all members of the same **Connection Object Class**. A device's Connection Object(s) is/are simply configured to consume data, produce data or consume and produce data. Those connection objects have absolutely no idea which of the three architectures they are operating within. Given these facts it should be no surprise that all three of these architectures can exist on the same network at the same time.

Conclusion:

As we have seen DeviceNet Connection Objects can be configured to produce data, consume data or both produce and consume data. Also, Connection Objects capable of data production can be triggered to do so by any one of the following three methods:

- Cyclic timer expiration
- A change of state of the communicated data
- Application specific trigger

In addition we have seen that these connection objects can be used to create the following four basic connection types:

- Unacknowledged Point-to-Point
- Acknowledged Point-to-Point
- Unacknowledged Multicast
- Acknowledged Multicast

Through the use of these connection types, data can be communicated throughout any one of the discussed control system architectures. Finally we saw that the system communication model was independent of the system architecture in which it was employed. DeviceNet's architecture independence provides the control system engineer with the flexibility to design control schemes using any system architecture or, if desired, a hybrid of two or more system architectures. While designing hybrid architectures may not become common, the ability to do so should prove to be a comfort to those who must provide control solutions for today while still maintaining a migration path to tomorrow.