Tech Note: Introduction to the Producer/Consumer Model

John Socha-Leialoha

1 Overview

In this document I describe a form of the producer/consumer (P/C) model, which has been around for many years and has been used in factory automation and other applications involving many devices communicating with other devices. The form presented here is simpler than many of the existing implementations, which makes it easier to implement and use.

Before describing the P/C model, it might be helpful to review some of the options for controlling how messages are sent between devices.

2 Source/Destination Approach

Traditional networking uses some form of source/destination addressing where one device (the source) sends data (a message) to another device (the destination). Often both the source and destination are identified in the data packet sent, which allows the destination device to send a reply directly back to the source (see Figure 1).

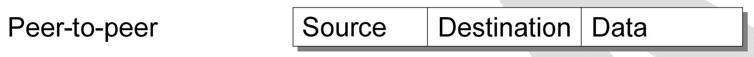

Figure 1: Traditional peer-to-peer message contain a source and destination address as well as the data being sent.

When no reply is required, these messages can be shortened slightly be providing just the destination address. In this form the messages are really commands being sent to a specific device.

2.1 Source Only—Broadcast

The case when only a source is present can handle the one-to-many situation. In this case, devices can listen for messages from a specific source. Typically the receiving device must be able to interpret extra data from the source to know what it should do. For example, a device that sends out state change information would send the new state in the data portion of the packet. This type of message allows one-to-many connections—one device, the source, can send messages that will be received by any number of devices.

Of course, all devices have to agree on the format of the data before they can work together properly.

2.2 Destination Only

Using only the destination in a message provides the standard command approach. Messages are sent to a specific device and the meaning of those messages is embodied in the data sent along with the message. The sending device needs to know the correct data format for the target device. This approach allows many-to-one connections—many different devices can send the same command to a single target device, the destination.

2.3 Source and Destination

The final case is the classic peer-to-peer case, which is a one-to-one connection only. One device, the source, sends a message to one other device, the destination.

2.4 Summary of Source/Destination Messages

The classic source/destination types of messages handle one-to-one, one-to-many, and many-to-one connections. However, they do a lousy job of handling the many-to-many connections. In addition, the source and destination devices must agree on the format of data sent with the message before they can be connected and work well together. The many-to-many case is where producer/consumer message have a real advantage, as explained in the next section.

Source/destination messages, however, are perfect for commands such as programming messages that really are intended for a single destination device.

3 Producer/Consumer Approach

In this section we'll look at yet another message model that differs from the classic source/destination model is some very important, and also subtle, ways. First, this approach allows many-to-many connections easily. Second, and perhaps more importantly, it allows connecting devices without any standards for data interchange format. In other words, you can connect very different types of devices that in the source/destination model might not be compatible. This interoperability is something not to be underestimated in networks with many devices interconnected, as is common on a model railroad.

DeviceNet (see references at end) uses a model called the producer/consumer model, which is also known as the publisher/subscriber model. In this model, messages are produced (broadcast) by devices and then consumed by other devices. These messages contain an identifier followed by optional data, as shown in Figure 2. There is no information whatsoever about either the source or destination of these messages. Likewise, in many cases the message consists of the identifier and **no data**.

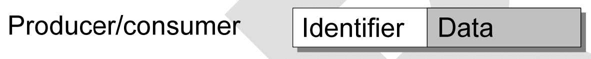

Figure 2: Producer/consumer messages contain no information about the source or destination of the message—just an identifier and optional data. Also, producer/consumer messages often have no data.

Devices in this scheme are all loosely coupled. Producers create and send messages without knowing anything about the devices that will consume those messages. Likewise, consumers receive and process messages without knowing which device sent the message.

These messages are very different from commands because they carry no information about what reaction a consumer should have to the message. A good way to look at this model is to think about cause and effect. In other words, something happens, such as the press of a button, and various other devices react.

3.1 Actions and Reactions

All of this works through actions and reactions (see Figure 3); actions produce messages and reactions consume messages. Each device can have a set of actions that are able to produce messages, as well as a set of reactions that can consume messages.

Actions are often triggered by a change of state or external stimulus. For example, pushing a button could trigger an "On" action and releasing the button could trigger an "Off" action.

Reactions, on the other hand, cause something to happen in the device. For example, telling a turnout controller to close will close the points on the turnout.

Device	Actions	Reactions	Values
Button	On Off		
Turnout Controller	Thrown Closed Moving	Throw Close Toggle	Invert
LED		On Off Blink	Blink rate Duty cycle
Servo		Position 1 Position 2 Position 3	Position 1 value Position 2 value Position 3 value

Figure 3: Here are some examples of different types of devices you might have.

3.2 Tying Actions to Reactions with Events

All that is missing is a way to have an action cause a reaction. For this we use the producer/consumer message, which is a message identified by an Event Number. These event messages just contain the Event Number (plus some header information). Event messages **do not** carry any state or address information.

3.2.1 A Simple Example

Let’s start with a simple example, with one-to-one connections. Imagine we have a SPDT, center off toggle switch that is connected to a device (inputs on a circuit board). These toggle switches have three positions, so the device can assign one action for each position, such as Up, Off, and Down. Whenever you move the toggle switch, it will “trigger” the action that corresponds to the new position of the toggle switch. Triggering an action doesn’t automatically send out a message. Instead, you must assign the action an Event Number so it knows what number to use in the message it sends.

Likewise, you might have a special turnout controller that can set a slip switch to three positions: Straight, Left, and Right. Each of these reactions can listen for a message (based on its Event Number), and upon receiving that message, perform the appropriate reaction. Again, you need to provide the reaction with an Event Number before it will respond to the message with that number.

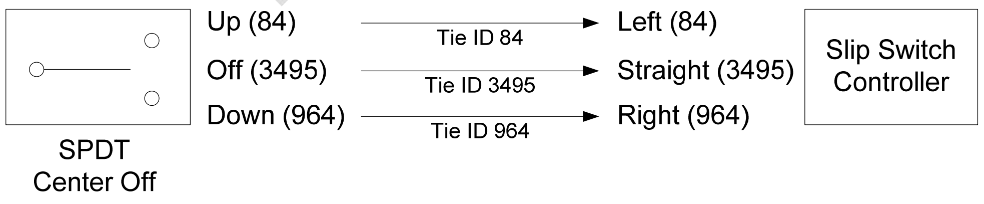

Figure 4: An example of a Single Pole, Double Throw, Center Of toggle switch attached to a device that produces messages, with another device that controls a slip switch.

Figure 4 shows three connections that have been created (how this is done is described later) that connect the three producer actions to three consumer reactions. We've told the toggle-switch device on the left to send out messages 84, 3495, and 964 for the actions Up, Off and Down, respectively. Likewise, we've told the slip switch controller to listen to messages 84, 3495, and 964 and change the slip switch to left, straight and right, respectively.

3.2.2 Setting up Events

Setting up events is done via programming. You need to tell the producer device which Event Number to send for an action, and you need to tell the consumer device to listen for that Event Number.

One question that often pops into people's mind is "how does the consumer know what to do with the state information sent by the producer?" For the most part, *there is no state information*, so this question isn't relevant. Instead, we use a different action for each state, as shown in the previous example.

This is subtle, so we want to say it again. In most cases, the producer sends no state information. Instead, the different states are represented by different actions.

The advantages of using different state actions over sending state data are subtle. First, it provides loose coupling. In other words, because you have multiple actions, you can tie an action that represents a specific state to a reaction. Pushing a button, for example, can cause a turnout to toggle. Or if you have a toggle switch instead of a push button, the "On" action could throw the turnout and the "Off" action could close the turnout. You have a lot of freedom to create complex interactions between devices without any state data because the state is embedded into multiple actions instead of a single message with state.

Another subtle advantage of the producer/consumer model is that Event Numbers are just numbers. There are several implications of this:

- You can have more than one device that *produces* messages with the same exact Event Number. For example, you might have two push buttons that trigger the same reaction.
- You can have more than one device that *consumes* messages with the same exact Event Number.
- When more than one device produces a specific Event Number, each of those devices can have a different action produce that Event Number. For example, in one case it might be an On action, and in another it might be a Pushed action.
- When more than one device consumes a specific Event Number, each device can perform a different reaction.

As you can see, the producer/consumer model is not a one-to-one relationship. Traditional source/destination messages are limited to one-to-one, many-to-one, or one-to-many. The producer/consumer model can handle all of these, but also the many-to-many case.

This approach is particularly useful when you have a number of different devices from different manufacturers that need to interact with each other in different ways. Noticeably absent is any specification for different types of devices. In fact, we went to great lengths to prevent having to specify all types of elements while still allowing complex many-to-many interactions between devices. At the same time we wanted to make such interactions relatively easy to configure.

As a brief side note, in rare exceptions actions also send extra data. A fast clock, for example, might have a single action that is triggered whenever the time changes, and the actual time is sent as extra data in the action. You could, if you wanted, tie the fast clock action to an LED that would toggle between on and off whenever it receives the fast clock message (although I'm not sure why you would want to).

3.2.3 A More Complicated Example

Now let's take a look at an example that is likely to be more common and is very difficult to setup using traditional commands without a separate controller, such as an external computer.

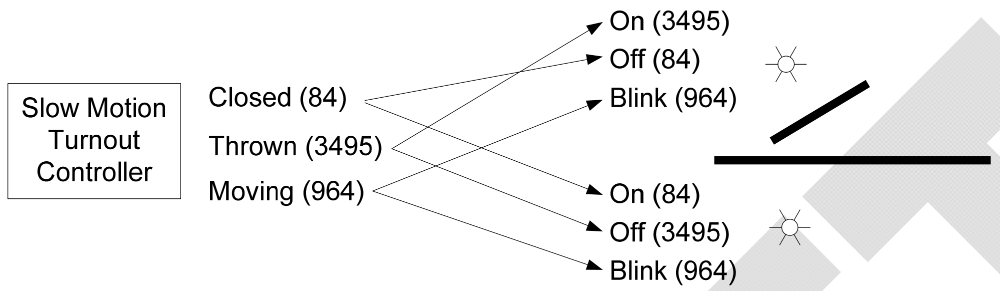

Figure 5: The two lights on a panel could show the actual position of a turnout, and blink while the turnout is changing positions.

In this example (see Figure 5) we have a controller for a slow motion switch machine. We're assuming that the controller knows when it's changing position and has a way to know when the turnout finishes moving. This might be controlling a slow motion switch machine that has contacts to indicate the actual position.

Next we have a fascia panel with two lights on the panel to indicate the position of the turnout. Just to make it interesting, we'll have these lights blink whenever the turnout is changing states.

You'll notice that the three actions on the turnout controller each produce a single message, 84, 2495, and 964. Each LED has three reactions it can take: turn on, off, or blink. When the turnout closes, the Closed action will send out message 84, which is then received by both LEDs. One will turn on, while the other turns off. Likewise, the Moving action sends out message 964, which is received by both LEDs' Blink reaction, causing both LEDs to blink.

3.3 Panel Example

The next, somewhat artificial, example you might find on a model railroad. Figure 6 shows a small yard on the layout, a fascia panel located at the yard, and a remote panel. The layout contains four turnout controllers (TC) to control the four turnouts for the yard.

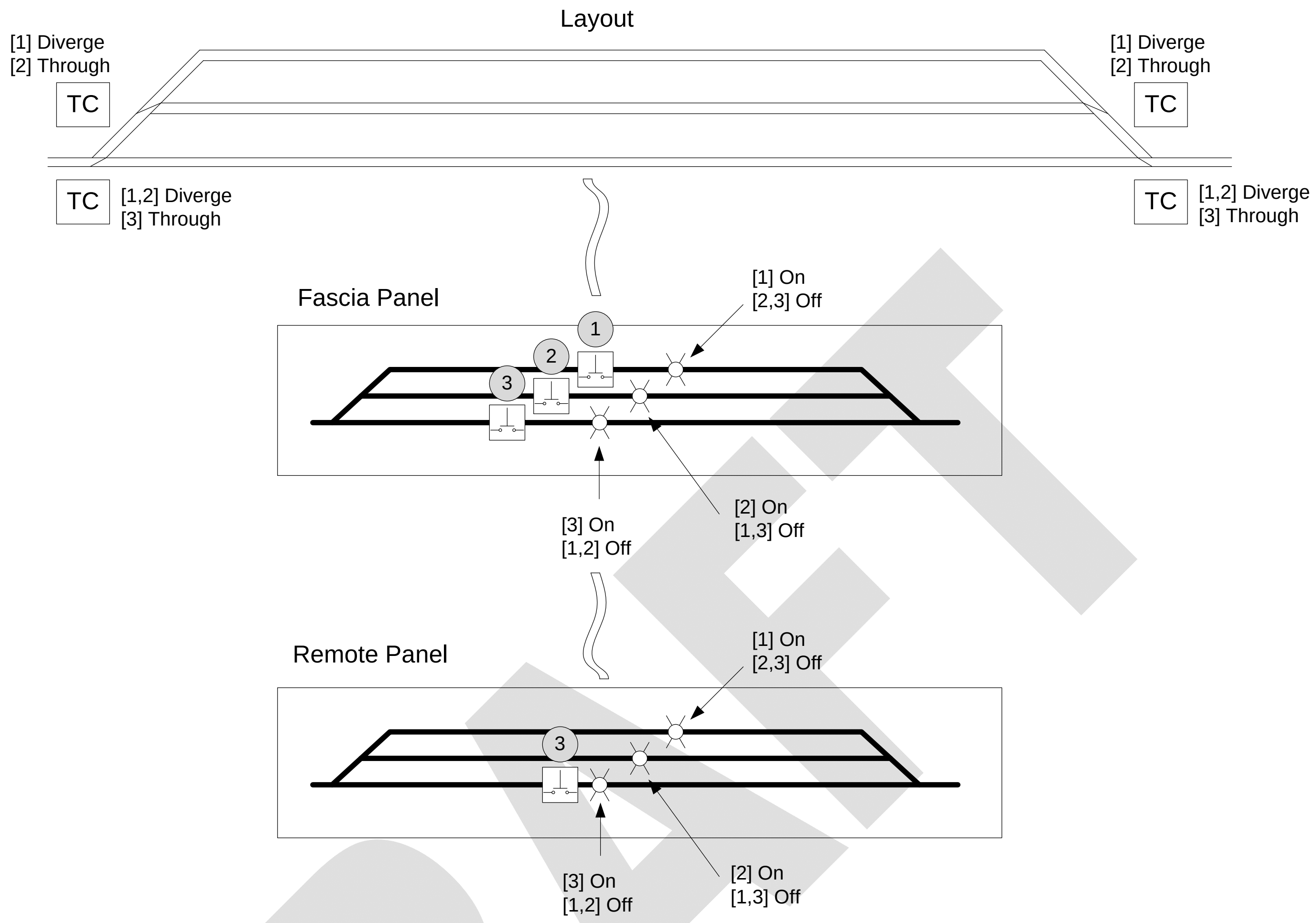

Figure 6: In this example, a small yard, a fascia panel, and a remote panel are all connected together using NMRAnet. Turnout controls are represented by TC. The fascia and remote panels have push buttons and lights. Push buttons generate messages to which the lights and turnout controllers respond.

The Fascia panel has three push buttons and three lights. Each push button will make one route the active route, setting the turnouts accordingly. One of the three lights will then be lit to show which route is set. The remote panel works in much the same way, except it only has one push button to align the turnouts for the mainline, leaving the other two routes under local control.

Now imagine how you would wire this up in a normal or DCC environment. It's not easy. Here is how it works with the producer/consumer model.

Each button in this example has been programmed with the Event Number to use when that button is pushed. The fascia panel's three buttons are assigned the numbers of 1, 2, and 3, as shown in the circles. The remote panel's single button also issues event message number 3 when you press this button so that it causes the exact same behavior as pressing the mainline button on the local panel.

When you press either of the "mainline" buttons, it sends event message 3 on the network. In this example, we've programmed the turnout controllers and the lights to listen for event message 3. When the lights receive event message 3, the two sets of yard lights will turn off, while the two mainline lights will turn on.

Only two of the turnout controllers, however, listen for event message 3 (since it doesn't matter which yard track is selected when the two mainline turnouts are lined for the main). These two turnout controllers will ensure the turnouts are lined for the main.

For each light and turnout controller in this example, you'll see a set of rules, like this:

[1] On

[2, 3] Off

These are very simple rules that say which event messages will cause each reaction. Event message 1 turns the light on, while either event message 2 or 3 will turn the light off.

3.4 Complicated Interaction Example

The final example shows the power of the producer/consumer model. Here we're fully utilizing the many-to-many case that is much harder to implement with the source/destination model.

In this example, we have a turnout controller that can be thrown or closed by either pressing a button or automatically by an engine entering a block. Additionally, we have two LEDs that show the actual position of the turnout.

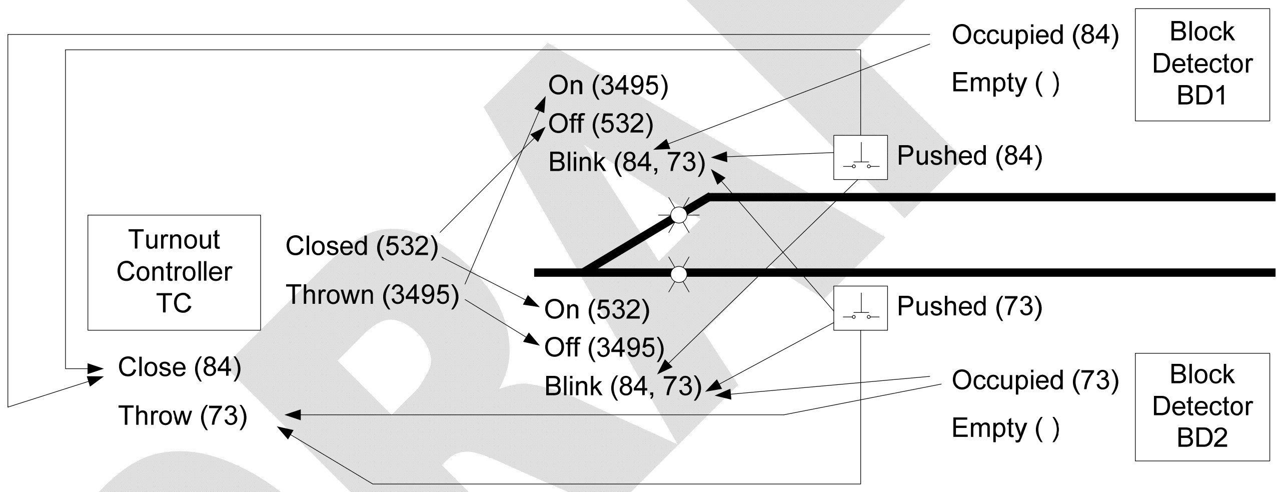

Figure 7: Here you can see the event connections that allow both push buttons and block detectors to activate a turnout.

We have a total of seven devices in this picture: one turnout controller, two LEDs, two push buttons, and two block detectors. Wiring all this up to work the way we want requires just four event numbers. Two of the reactions on the LEDs require responding to multiple events. The arrows in this diagram represent event connections between actions and reactions.

3.5 Bow Tie Diagrams

All the lines in Figure 7 can quickly become overwhelming. Fortunately, there is another notation, called bow tie notation, which makes it much easier to describe the events in the system. Figure 8 shows the four events. On the left are the actions that tie into each event and on the right are the reactions to each of these four events.

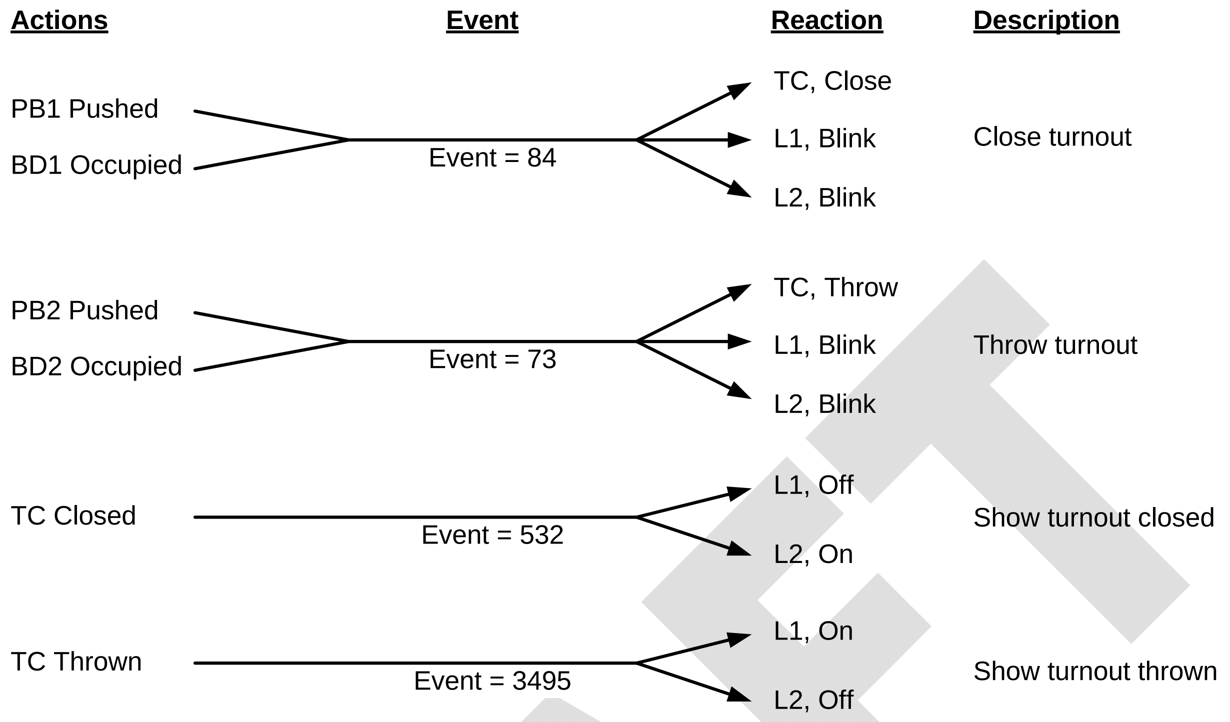

Figure 8: "Bow-tie" diagrams provide a simple way to show ties between actions and reactions using events.

Using this notation, it is fairly easy to describe rather complicated many-to-many interactions. In fact, you can envision a computer program providing a user interface that allows you to create and maintain these directly. Such a program could also hide the actual assignment of Event Numbers.

3.6 Advantages of the Producer/Consumer Model

You may have notice that the devices in this model are very simple. They have no rules engines, and yet you can create some fairly complex interactions. The "rules" in this case are actually distributed among the different devices in the network, and is reflected in the ties between actions and reactions.

Even more importantly, this model provides a very high level of interoperability between a wide variety of devices from different manufacturers. As long as each device has a set of actions and/or reactions, it can work with completely different types of devices from other manufacturers. You'll notice, for example, that we didn't define a standard input device, or a standard output device. In other systems these types of devices must be defined so you know how to interact with them. But in the producer/consumer model, the interactions are between actions and reactions without any knowledge of the device types involved. The only time you need to know about device types is when you setup events between devices.

3.7 Tying Actions to Reactions

Actions and reactions are tied together via an Event Number. You create this connection by assigning this Event Number to actions on one side and to reactions on the other side. Going back to the example in Figure 8, let's look at how you would setup the connections for Event Number 84 (the top bow tie). Here are the steps you would perform (in any order):

- In push button 1, set the Event Number to 84 for the Pushed action
- In Block Detector 1, set the Event Number to 84 for the Occupied action
- In Turnout Controller, add the Event Number 84 to the Close reaction
- In LED 1, add the Event Number 84 to the Blink reaction

- In LED 2, add the Event Number 84 to the Blink reaction

You may have noticed a subtle distinction we made between actions and reactions. You set the Event Number for actions, but you ***add*** the Event Number to a reaction. This difference is to allow a single reaction a device to respond to more than one Event Number.

4 References

“The Next Generation Networking Paradigm: Producer/Consumer Model,” http://www.dedicated-systems.com/magazine/00q1/2000q1_p026.pdf

“Networking From the Device to the Internet.” http://www.ab.com/networks/device_internet.html

“Controller Area Network – Introduction,” http://www.ixxat.com/can_introduction_en,7521,5873.html