BPMN and the Business Process Expert, Part 6: Choreography and Multi-Pool Processes

Summary: In addition to describing the internal process orchestration, or control flow, BPMN can represent choreography, the message exchange between processes. In the real world, an end-to-end business process may be composed of multiple BPMN processes interacting through choreography. This discussion sheds light on BPMN's most subtle concepts: just what is a "process" in BPMN, and what does a "pool" really represent? Last of six parts.

Author: Bruce Silver

Company: Bruce Silver Associates

Created on: 31 December 2007

Author Bio

Dr Bruce Silver is an independent industry analyst and consultant focused on business process management software. He provides training on process modeling with BPMN through BPMessentials.com, the BPM Institute, and Gartner conferences, and is the author of The BPMS Report series of product evaluations available from the BPM Institute.

Now that we've explored the basics of BPMN, let's return to the most basic concept of all: What is a process? More specifically, what is a *single* BPMN process, as opposed to *multiple* processes linked by message flow choreography in a single business process diagram (BPD)? We know that a BPMN process is confined to a *pool*, and a BPD can contain multiple pools, but what does a pool really signify?

The BPMN spec does not explain these things very well, and many BPMN tools that front-end an execution runtime do not even support BPDs containing multiple pools. Moreover, in those that do, most use those extra pools to represent only external entities that invoke your process and receive responses from it, not other parts of your own internal business process. Nevertheless, as you begin to apply BPMN to model real-world end-to-end processes, you will find that sometimes you cannot confine them to a single pool. Thus to be a true business process expert, you need a deeper understanding of processes and pools.

You frequently hear that a pool in a business process diagram represents an organization. That is incorrect. A pool is simply a container for a BPMN process. If your BPD does not have more than one pool, the pool containing your process might not even be drawn… but it is always there. If your diagram shows choreography between your process and an external process, a requesting client or invoked service provider, often the pool names may indicate the organization behind each process, but the pool actually represents the process, not the organization.

A BPMN process, or pool, represents a domain of control. With that process or pool, BPMN describes the orchestration, or flow of control, from one activity to the next. In a B2B process, if you are the seller, you do not control the buyer's process, just your own. In fact, you typically do not even know the precise orchestration of the buyer's process. If you want to show that pool in your diagram, you may declare it an *abstract process* or "black-box" pool, empty inside, with choreography drawn to the pool boundary. Alternatively, you may want to show activities representing specific touch points inside the buyer's pool. In that case you can declare it a *collaboration process* or "gray-box" pool, meaning those activities do not represent the buyer's detailed orchestration. Your own *private process* or "white-box" pool is the one that actually models actual orchestration within a single domain of control.

But sometimes your internal end-to-end process requires multiple private processes, all within your control. So there is more to defining a BPMN process than domain of control. To understand it, you need to think about the *lifetime* of a process instance, and what the instance represents. When the process starts, an instance of it is created in an initial state, and when the process completes, the instance is left in some final processed state. In between, the state of the process instance depends on all the activities that have completed up to that point.

It is helpful to think concretely about what the instance represents. For example, in an order-to-cash process, the instance typically represents an order. When the process is instantiated, the order state is simply *received*, and upon completion it could be *fulfilled-and-paid*, or *rejected*, or possibly *cancelled*, or some other end state (Figure 1).

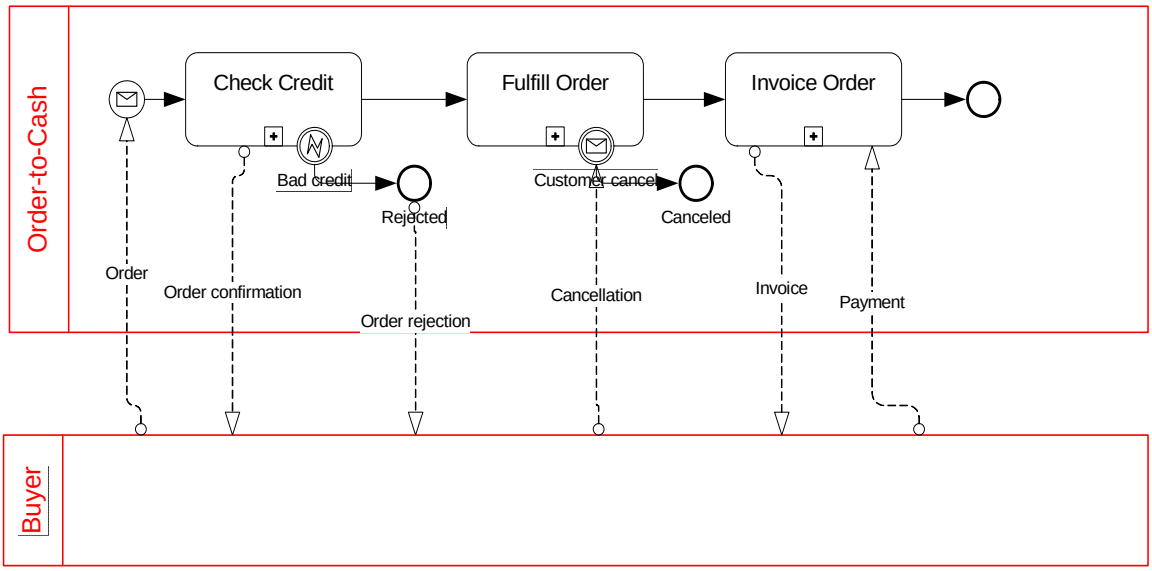

Figure 1. Simple order-to-cash process in one pool

Let's say the order contains a list of items, and some of them are out of stock. The in-stock items will be shipped and invoiced right away, but the others must be backordered and invoiced separately. Figure 2 shows the backorder within the original BPMN process. Figure 3 shows the backorder as a separate process invoked asynchronously ("fire-and-forget") from the original process. Let's look at the difference.

There is no "correct" solution. In Figure 2, the order instance is not complete until all parts of it, including the backordered items, have been fulfilled and paid. Who knows how long that could take, but the entire order instance remains incomplete until the backorder is done.

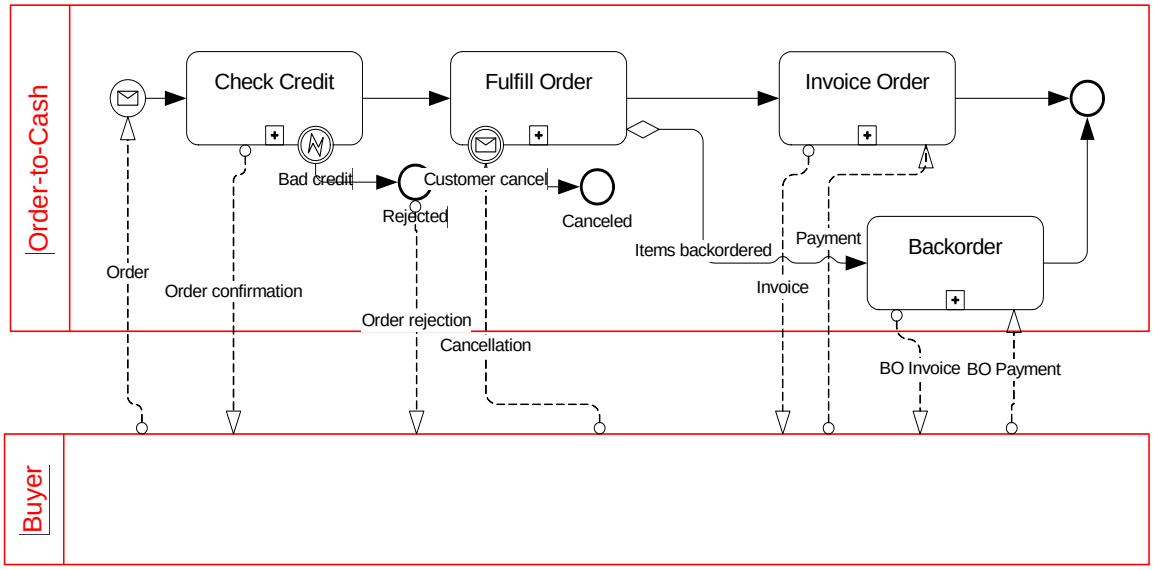

Figure 2. Handling backordered items within the order-to-cash pool

In Figure 3, the main process just handles normal order processing, while the exceptions – the backorders – are instantiated in a separate process. A message event – a signal to another pool – is thrown by an end event in the main process and caught by a start event in the backorder pool. This allows more flexibility, since in principle a separate instance of the backorder process could be created for each item not in stock, and each could proceed on its own timeline without affecting the others.

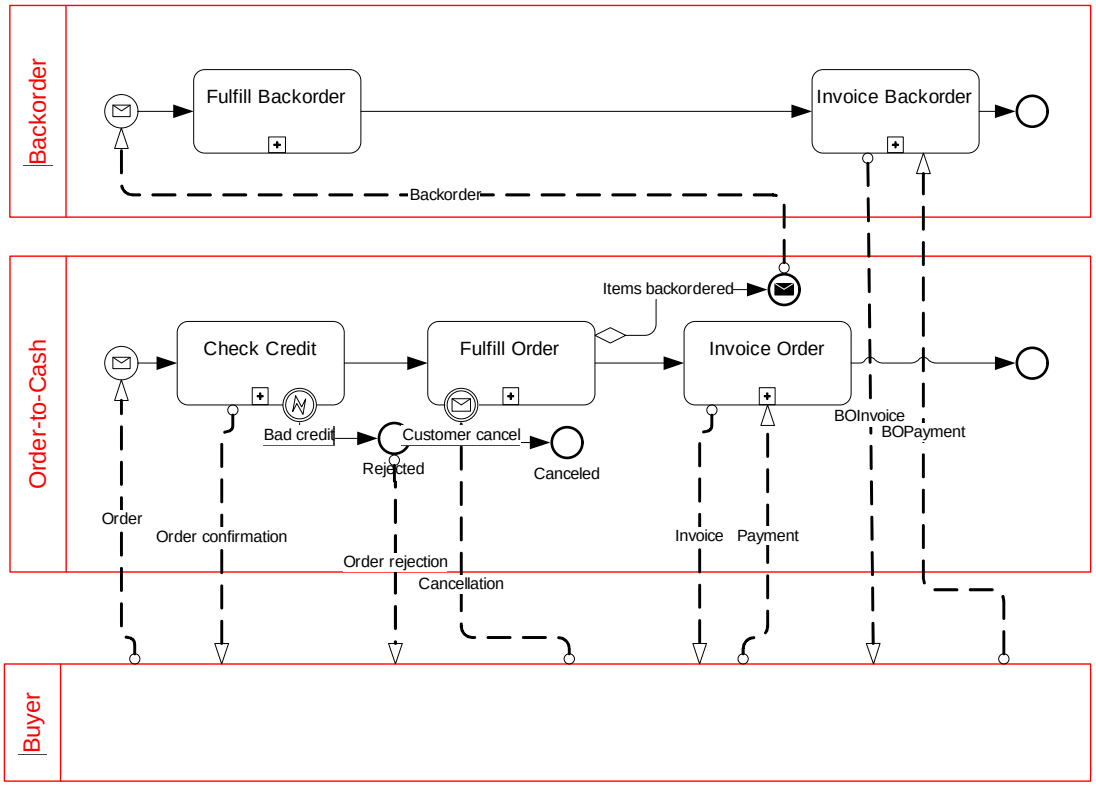

Figure 3. Handling backordered items in a separate pool

Notice another thing about both Figure 2 and Figure 3. Each payment message flow corresponds to a specific invoice message flow. In some cases that may be how it works, but instances of payments might not have a one-to-one correspondence to instances of invoices… and instances of invoices might not always correspond one-to-one with orders. Remember in our order-to-cash process an instance always means an order, but some of its activities, as they are actually performed, may deal with just part of an order or perhaps span multiple orders. This mismatch of instances is another reason for using multi-pool processes.

In the real world, an order handling process does not typically directly send invoices and receive payments from customers. In some cases, customers are invoiced monthly and pay monthly as well. For now, assume the invoice is sent directly from the order process but payment processing is separate, linked to the end-to-end order-to-cash business process via choreography. Let's see how that works in BPMN.

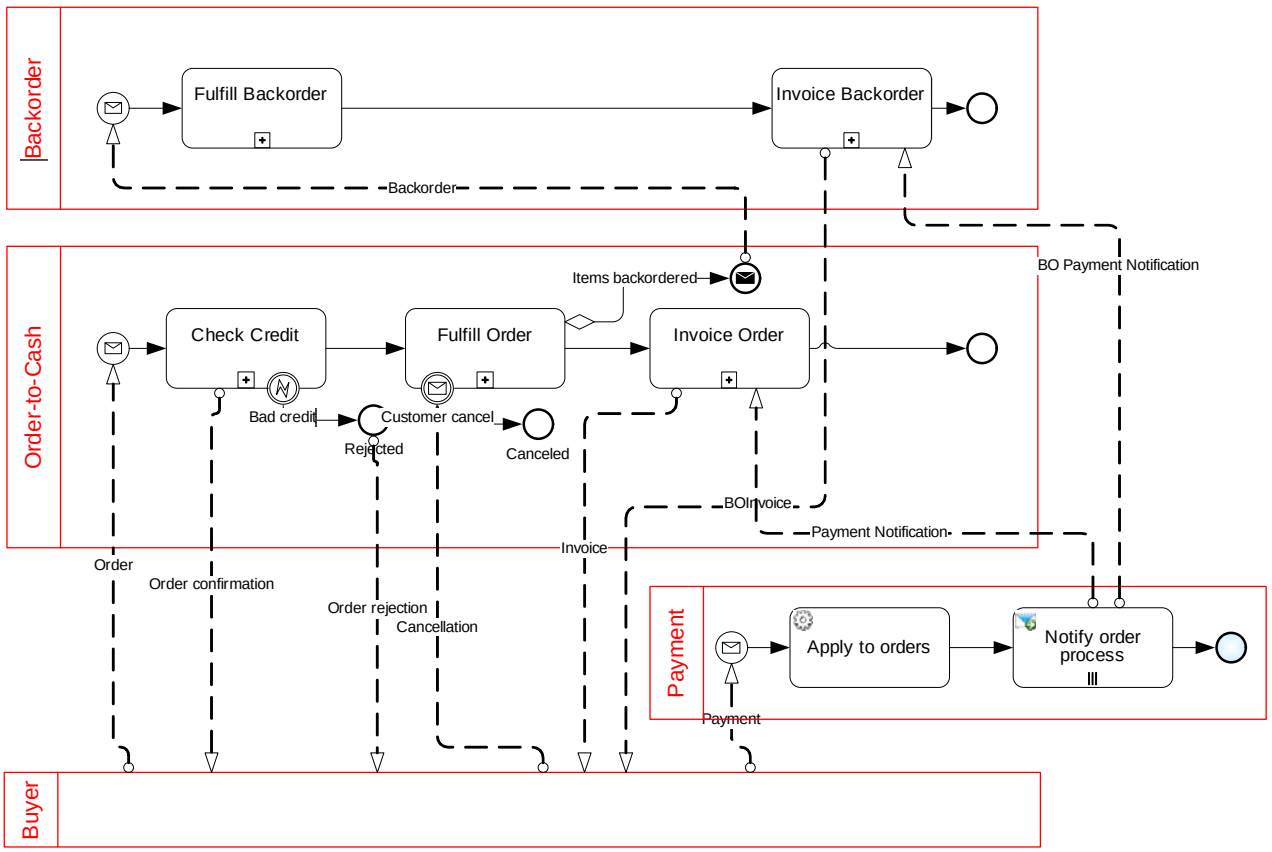

Figure 4. Payment processing as a separate pool

Figure 4 illustrates the most common reason for multi-pool processes: an instance of one process does not have one-to-one correspondence with an instance of the other. Here each payment from the buyer instantiates a payment process, which applies the payment against outstanding invoices, corresponding to orders. When payment for a particular order is complete, a *notification* is sent from the payment process to the corresponding order process. Inside the Invoice Order subprocess of Order-to-Cash, for example, a message intermediate event could wait for the payment notification message.

This is an example of a process modeling problem that frequently vexes students in my BPMN training classes, what I call the 1:N problem. N instances of some process fragment

relate to 1 instance of the end-to-end process. Sometimes you can solve this problem with multi-instance activities within a single pool, but multi-pool processes enable a better solution.

Figure 5 is a student submission for an exercise intended to show an employee hiring process. Can you see what's wrong with it?

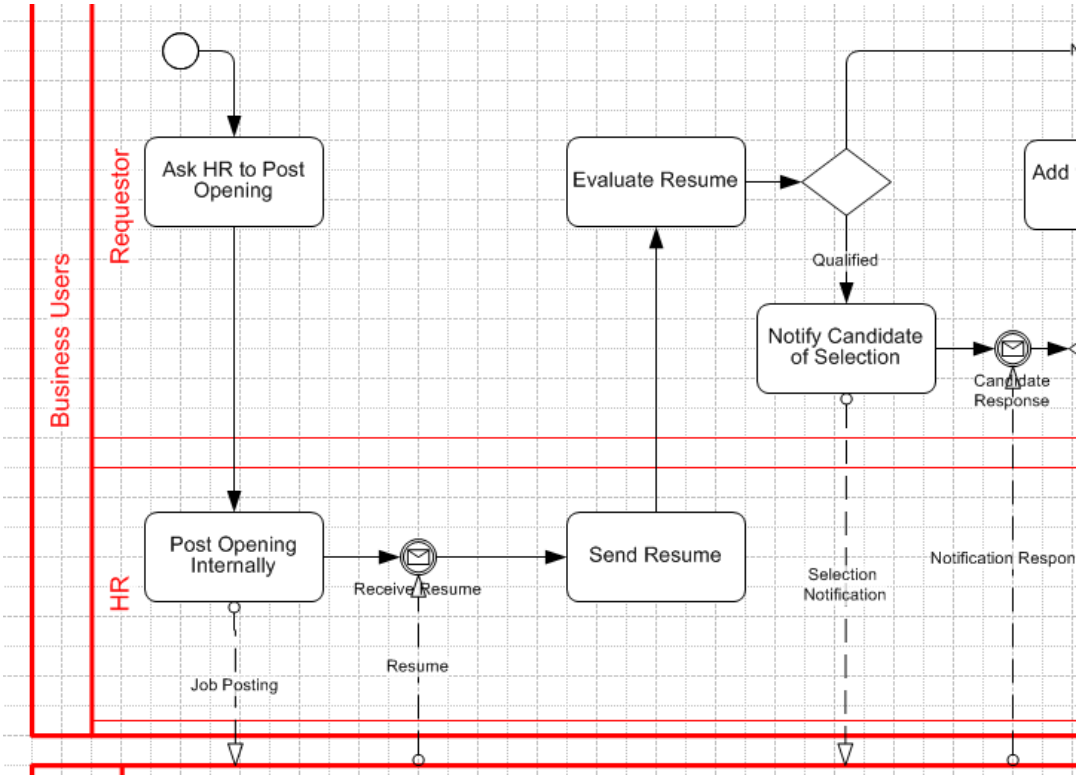

Figure 5. Student example exhibiting the 1:N problem

After the job is posted, the process waits to receive resumes from applicants, and then processes them. But as drawn, this model processes only one resume, the first one received. The student intended that each resume message flow would be received by the message event, but that's not the way BPMN works. Once the instance has passed that event, it cannot receive additional resumes. Here the overall process instance represents a job, but for a portion of the process, this diagram represents a single candidate. This is the 1:N problem.

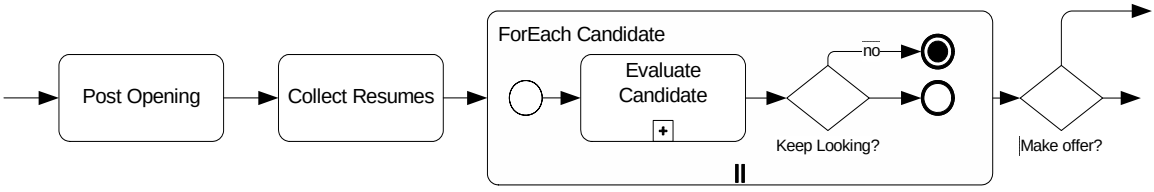

Figure 6. Solution to 1:N problem using multi-instance activities

Figure 6 illustrates a way to model this "legally" in a single pool using multi-instance activities. After the opening is posted, an activity collects resumes for some time period. After that no more resumes can be submitted. Then each candidate in the batch is processed individually using an MI activity. You don't even have to complete the whole batch; if you

find the right candidate, you can cancel other instances of the MI activity using the Terminate end event.

This model, while easy for students to understand, has some drawbacks. For one thing, all of the resumes must be collected in advance, and that activity must be complete before evaluating any of the candidates can begin. With multi-instance activities, you need to have all N instances available in advance. But in my experience, after you post the job, you want to begin reviewing applications as they come in, so receiving and processing applicants are actually concurrent activities. You can do this using multi-pool processes.

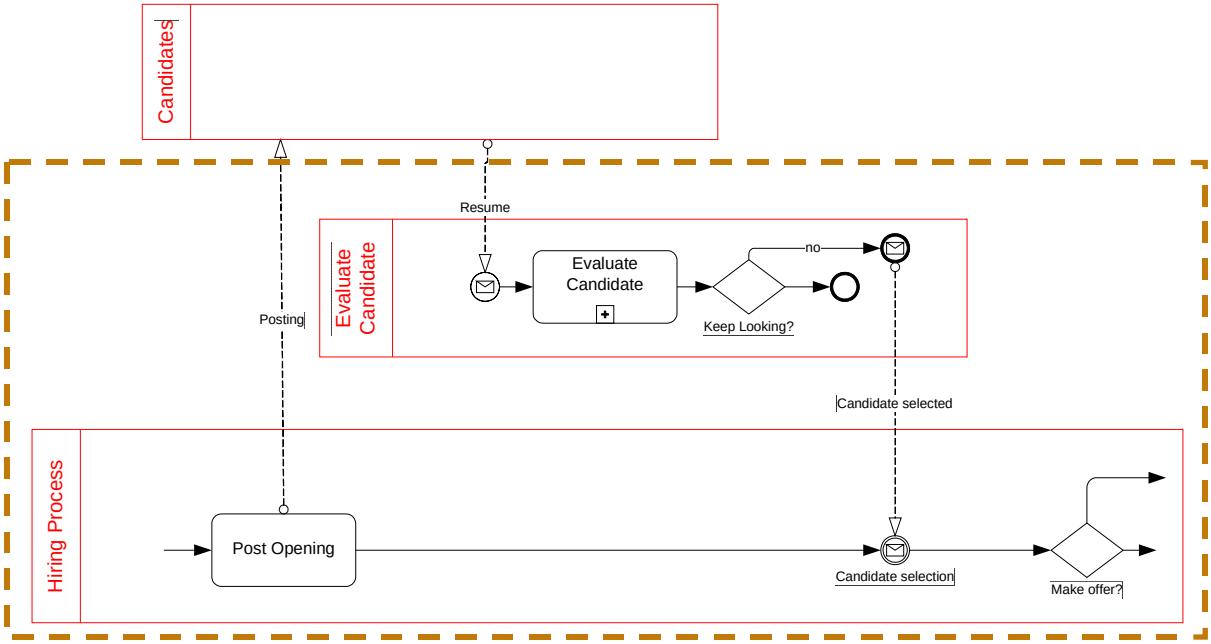

Figure 7. Multi-pool processes provide a better way to handle the 1:N problem

In Figure 7, candidate resumes are not received by the Hiring Process pool, but by a separate pool, Evaluate Candidate, in which an instance represents a single applicant. Each resume submitted creates an instance of that pool, so you don't need to know how many there are, and the start and end times of each instance are independent both of the main process and of each other, as well. The main Hiring Process pool is not directly evaluating candidates, but simply waiting for notification that a candidate has been selected. In BPMN terms, it is waiting for a message signal issued from Evaluate Candidate when that has occurred.

Note the Hiring Process and Evaluate Candidate pools both represent the same organization – maybe even the same individual participants. They are simply distinct BPMN processes, linked by choreography in a single business process.

These examples hopefully shed some light on the question, hard to answer directly, what is a BPMN process? A BPMN process has an identifiable instance, a well-defined beginning and end. However, that process may rely on activities in which the activity instance has no definite one-to-one correspondence with the process instance, or no definite start or end time relationship with the process. In such cases, those activities may be part of a separate BPMN process, linked to it via message flow choreography.

Multiple internal, or private, processes can be linked this way to represent an end-to-end business process. These processes may have independent start events and operate as peers in the BPD, not strictly nested or chained together. Correlating appropriate instances of each

process – implementation detail not always visible in the BPMN diagram – is critical for executing such a process.

In this series of articles, we've seen BPMN's ability to represent end-to-end business processes in diagrams that business people can understand, yet which retain remarkable precision and expressive power. Unlike traditional process modeling notations, BPMN puts events and exception handling right in the diagram itself, without requiring specification, or even knowledge, of the technical implementation. The combination of this business-friendly "abstract" representation with precise orchestration semantics lets BPMN process models serve as the foundation of executable process implementations, with implementation properties layered on top of the model. These implementation properties are added by IT, often in direct collaboration with business, and leveraging a common underlying model. This type of collaborative approach is essential if BPM is to realize its promise of improved agility and responsiveness to changing business needs.

If BPMN is the "language" of this emerging collaboration, the Business Process Expert is the agent of change. Knowing how to use BPMN to model processes correctly and effectively has become the critical skill for all BPXs to master.

Bruce Silver