

BPM 2.0

By Ismael Chang Ghalimi, CEO, Intalio

Table of Contents

Preface.....	3
Introduction.....	4
Used by Process Analysts.....	5
Starting with a Complete BPMS.....	7
One Single Tool in Eclipse.....	9
Loved by ABAP, PHP and VB Folks.....	11
BPEL.....	13
BPMN.....	16
BPMN Designer.....	18
Zero Code.....	20
One Click Deploy.....	22
Generating Web Services on-the-fly.....	24
Interpreting BPEL Code Natively.....	26
Web 2.0 User Interface.....	28
Rule Engine Included.....	30
Real-Time BAM Included.....	32
Native Process Simulation.....	34
Dynamic Process Optimization.....	36
Open Source Process Engine.....	38
Get Started Today, Free of Charge.....	40
About the Author.....	42

Preface

This document is a compilation of 18 weekly articles posted on the [IT|Redux](#) blog between March 13, 2006, and July 10, 2006. Original posts have been slightly edited in order to fit within the format of this compilation. Additional references and user comments are available online.

This document was edited using [ThinkFree](#) Office 2.0 productivity suite.

Introduction

Six years ago, I wrote the [first white paper on BPMS](#). It was one of the seminal publications that helped define the concept for BPM and start a new industry. The three letter acronym, which we borrowed from musicians, became an instant sensation, successful beyond any expectations we could have had at the time. Too successful some would say.

Today, the BPM moniker is used to describe anything from legacy workflow products to business rule engines, flowchart diagramming tools, Java code generators, or even business process reengineering consultancy services. This confusion, perpetuated by software vendors and industry analysts alike, serves two main purposes: it allows any vendor who can show boxes and arrows in its product to keep selling its gear, while letting any analyst who can compile a list of the aforementioned vendors to sell its luminary services to herds of utterly confused end users.

Customers I talk to are asking for a change. They've tried the first version of BPM, did not find what they were looking for, and are wondering if there is anything else worth trying out. The good news: there is, I call it BPM 2.0—a term originally coined by my good friend Bruce Silver, and it's available now. The bad news: the definition I give for BPM 2.0 is a radical one, it leaves no place to hide, and most vendors won't like it. But guess what? I am more interested in making customers happy than letting other vendors sleep well at night, especially when they happen to be my competitors. So here we go, welcome to BPM 2.0!

Used by Process Analysts

vs. Marketed to Business Analysts

Let's start by debunking the biggest lie about BPM, which is that business analysts could use a BPM tool to model & deploy an executable business process. Even though that might be true for the simplest document-centric workflow processes, such as when a business analyst specifies the reviewing process of issuing press releases on a website, it breaks as soon as the process involves transactions with any kind of back office system, for two main reasons: first, last time I checked, no IT guy is ever going to open a port onto the corporate ERP system for a business analyst to mess with; second, the said business analyst does not want to be the one the CEO calls in the middle of the night if the aforementioned ERP system cannot record new purchase orders. Conclusion: BPM 2.0 is not for non-technical business analysts. Never should have been, never will, and nobody should care. Instead, BPM 2.0 is for process analysts who are articulate enough to talk to business folks, yet technical enough to understand the difference between a do-while loop and a for-each statement. We will not bridge the business-IT divide by empowering business analysts to get rid of IT people. Instead, we'll just let more technical process analysts understand business requirements and implement them directly into the process, while leveraging existing IT systems. Neither top-down nor bottoms-up, it's a middle-out approach, and it's the only one that makes the gap any narrower.

But who is a process analyst? A simple test is to ask a candidate if she understands the differences between a do-while, a while-do, and a for-each loop. If she does—and can explain it to a business analyst who could not explain what a loop is at the first place—you found the person you were looking for. You will find her among the 8 million Visual Basic programmers, the 3 million PHP fans, the million PL/SQL folks, and the half a million or so ABAP guys and gals. If you compare that to the 2 million people who can write Java code today, that's a pretty big group. If you add to the mix the folks who understand HTML, you've got a pool of more than 20 million people you can draw from.

And if you want to understand why the mastery of Java and J2EE should not be a pre-requisite to the use of a BPMS, just read the [syllabus](#) for the course on jBPM offered by JBoss:

“The student must have previous experience developing an Hibernate application. The student must know how to configure a simple SessionFactory for Hibernate, utilize a Hibernate Session and transactional demarcation and how to perform basic queries on Hibernate objects.”

To me, it’s pretty much a show stopper. BPM 2.0 does not stop there though.

 [Original Post](#)

Starting with a Complete BPMS

vs. Starting with a Process Modeling Tool

Because BPM was originally marketed to business analysts, vendors thought that it would be a good idea to start with the only tool business analyst could use, namely some kind of flowchart diagramming tool. Problem is, that's exactly what customers did, but they did nothing else beyond that, for a very simple reason: once a business analyst has diagrammed a process with a tool that does not enforce any rule that would make the process executable, there is no way to make that process executable later. All that work goes to waste and the business analyst feels she's been cheated.

If you want to try this BPM 2.0 thing out, my advice is "don't buy a process modeling tool". Instead, get yourself a complete BPMS and start building executable processes from day one. Some would call it Agile Development for business processes. I call it BPM that works.

But what is a complete BPMS? To me, a BPMS must support both process design and process execution, therefore a simple process modeling tool with process simulation capabilities does not qualify as a BPM 2.0 product. Also, a BPMS must support both web service orchestration and human workflow interactions. You can think of it as two sides of the same coin, one facing back-end IT systems, the other facing front-end human beings. As a result, and at bare minimum, a complete BPMS has three main components: a process design tool, a process execution runtime, and a workflow user interface.

If you want a more complete BPMS, three additional capabilities are needed: support for complex business rules, Business Activity Monitoring (BAM), and a way to manage the versioning of documents that are attached to process instances. It does not really matter if these capabilities are offered natively by the core BPMS, or are provided by external components, as long as the life cycle of business processes is preserved, with **zero code** and **one-click deploy**.

And if you want to be really fancy, three more things can be added: a complete Enterprise Service Bus (ESB), a metadata repository, and a business intelligence suite that will help you slice and dice the data coming out of the BAM infrastructure. Once you reach that level of sophistication, you have a complete BPMS that can be used to manage virtually any kind of business processes, within the most complex environments.

Now, let's take a look at how [Intalio](#) is building such a thing. First, we developed a [process design tool](#) (using [Eclipse](#) as underlying workbench), a [process server](#), and a [workflow suite](#). Second, we picked [Corticon](#), [Celequest](#), and [Alfresco](#) as preferred partners for business rules, business activity monitoring, and content management respectively. Integration with these products is done through our [Demand Driven Development](#) program, and all three products come for free as part of our [Community Edition](#). Third, we selected [ServiceMix](#), [Infravio](#), and [Pentaho](#) for the Enterprise Service Bus, metadata-repository, and business intelligence suite respectively. At present time, we are still working out the details of integrating with Infravio and Pentaho. Integration with ServiceMix was completed two weeks ago, which gave us the first JBI-compliant BPM engine. More recently, we also added support for [Apache AXIS 2.0](#).

If you're looking for a complete BPMS, [give Intalio a try](#).

 [Original Post](#)

One Single Tool in Eclipse

vs. Multiple Tools from Multiple Vendors

Not long ago, going from a process map to deployed code used to take up to seven tools: one for modeling the process at a business level, an other to describe technical details, a third to build connectors to external systems, a fourth to map data in and out, a fifth to specify business rules, a sixth to design workflow user interfaces, and a seventh to deploy all the code on a collection of proprietary runtimes components. They all required different kinds of expertise, ran in different environments, used different languages, lost information when going from one to the other, and made the whole thing so complex than no end-user could actually use them without the outrageously expensive consulting services of a software vendor that got all the pieces through multiple acquisitions rather than building them from scratch with a very clear architecture in mind.

BPM 2.0 puts everything you need within one tool, and that tool sits on top of Eclipse. With Oracle and SAP now supporting Eclipse, no other integrated development environment—beside Microsoft’s Visual Studio—will matter anymore, so get along with it and demand that your BPM 2.0 tool natively runs in Eclipse.

But why does Eclipse matter? My definition for BPM 2.0 states that a BPMS should be built around **one single tool in Eclipse**, and this radical statement deserves further explanation, as was illustrated by some of the discussions that followed the **original post on BPM 2.0**.

Integrated Development Environments (IDEs) are very much like Operating System, in the sense that they provide the foundation upon which software vendors can build their tools, without having to re-invent the wheel all the time. And much like Operating Systems, they tend to be the victim of commoditization and consolidation, rather sooner than later. Oracle became one of Eclipse’s strongest supporters, Borland has decided to sell its JBuilder group, and NetBeans is not exactly driving developers to Sun’s platform in droves.

This pretty much leaves two contenders on the ring: [Eclipse](#) and [Microsoft Visual Studio](#). But because Microsoft does not like to leave much room to its competitors, the vast majority of BPMS vendors have adopted Java as a development platform, and this is where Eclipse comes into play. In other words, if you're Microsoft, your BPMS will be built around Visual Studio, but if you're not, Eclipse is the way to go.

The main benefit of using a standard IDE for developing the tools that are part of a BPMS offering is that you can leverage existing components that have been developed by third-party vendors. For example, [Intalio](#) is using a rule editor from [Corticon](#), a KPI designer from [Celequest](#), and a WYSIWYG XForms editor developed with [Orbeon](#), all of which are available as Eclipse plugins. Also, because Eclipse provides support for the most popular source control systems, integrating with the customer's application lifecycle management infrastructure is greatly simplified, and the BPMS vendor gets all this essentially for free.

That being said, using Eclipse instead of a proprietary framework creates a couple of challenges. First, Eclipse is a very complex platform that makes extensive use of advanced technologies such as [EMF](#) and [GEF](#). The learning curve for these is pretty steep, and the productivity of an average Java Swing developer converting to Eclipse can be significantly reduced for the first three to six months. Second, because Eclipse was originally developed for software engineers, it makes everything more complex than it should be for your average [process designer](#). As a result, a BPMS vendor adopting Eclipse as underlying IDE must make sure to hide this complexity under simpler user interfaces. This is not an easy task, but the benefits you can gain by running on top of Eclipse largely compensate for the migration effort. Thank you IBM for such a fine piece of software!

 [Original Post](#)

Loved by ABAP, PHP and VB Folks

vs. Usable by J2EE Experts Only

BPM products of the first generation required expertise with J2EE, or even worse, proprietary scripting languages, as if the world needed yet another programming language. As much as I like J2EE for what it gives me as a software vendor, it's a freakishly complex set of specifications that are out of reach for most IT people. Object-oriented programming is extremely powerful, but like it or not, most programmers do not understand it, or at least would rather use something simpler like PHP or Visual Basic. There are 2 million Java programmers out there, and I reckon that only a fraction of them can write EJB components. Even less know how to combine EJB with JMS, Servlets and Message-Driven Beans. Compare that to the 3 million PHP coders and the 8 million VB developers out there, and you'll start getting the picture. BPM 2.0 is not for the J2EE gurus, or at least not limited to them. BPM 2.0 targets process analysts who can read a BPMN diagram, understand the tree-view representation of an XML Schema, and drag-and-drop form widgets onto a canvas. If you think you can do that without too much effort, then BPM 2.0 will work for you.

But what is wrong with J2EE? Java—and its enterprise offspring J2EE—are extremely powerful technologies. Problem is, with power comes complexity, and J2EE is a perfect example for it. The Java Community Process (JCP) website lists 320 [Java Specification Requests \(JSRs\)](#), and the [Java Enterprise Edition 5 Beta API Specification](#) has 1438 classes. Whichever way you look at it, Java grew big and complex over time, and things are not getting any better.

J2EE was supposed to bring portability for applications developed on top of an application server, but reality is that beyond the most simple ones that only take advantage of a Servlet container, you cannot port applications from one application server to another, especially if they make use of Enterprise Java Beans (EJB) components.

The **EJB 3.0** component model is promising to change that, and it might. Unfortunately, when a problem is fixed here, a new one pops up there. We experienced this recently when we tried to port our process server on top of **Apache Geronimo**. Everything worked fine until we tried to package all the components together—process server, workflow services, XForms engine, etc.—then it broke. Somehow, one component was turning a shared **Log4j** service off, and we had no easy way of isolating which component was at fault. This vexing problem, alongside a dozen similar roadblocks, kept a team of three highly-talented Java developers busy for the best part of last quarter. If a software company like Intalio, which business is to develop software, is facing this kind of challenges, how is it for end-users whose primary business is selling widgets or processing financial transactions? Well, as far as I can tell from many discussions with customers, it's not that great either...

Your average developer cannot deal with this level of complexity. Not that she is not smart enough, but she's got better things to do, like taking care of business for example. And what is true for J2EE is also true for collateral technologies, such as Eclipse for example—have you ever tried to use EMF and GEF? For me, **BPM 2.0** is all about offering a quantum leap in abstraction that will allow your average developer—ABAP, PHP and VB folks—to bypass this complexity and focus on higher levels of the stack, namely processes, rules, and interfaces. To a large extent, these developers have felt left out with the exploding complexity of J2EE and .Net, and BPM 2.0 gives them the tools to be productive again, with a vengeance. That's precisely why they like it so much.

 [Original Post](#)

BPEL

vs. *BPEL, BPML, WSFL, XLANG, XPD*

In the early days, there was no standard for executable processes. XLANG was on the drawing board, WSFL did not exist, and the workflow guys had produced nothing more than a set of utterly useless interfaces. Then BPMI.org released the BPML specification, which forced Microsoft and IBM to abandon XLANG and WSFL respectively. For political reasons, Microsoft and IBM decided to write their own specification, rather than adopting BPML, which led to the release of WS-BPEL, a year after the first commercial implementation of the BPML specification was deployed into production. Three years of sterile public discussions followed, until BPMI.org merged with the OMG and finally decided to drop BPML in favor of BPEL. In short, no real standard was available for the last six years, which contributed to slowing down the adoption of BPM.

Things are a little bit different today. BPEL has won, BPEL 2.0 is a good enough specification for customers to build mission-critical processes with, and all the big vendors have adopted it. BPM 2.0 works because of BPEL, much like relational databases work because of SQL. Process analysts should not really care about it, for they won't have to write a single line of BPEL code if they pick the right tool, but BPEL is like the DNA of your process, it's the standard that everything else gets built around. So if you're being told that BPEL does not matter, or that the product you're considering buying will support BPEL next year, don't be fooled: you're about to buy a very expensive piece of proprietary software that you will have to get rid of sooner than you think, so don't make the same mistake that early adopters made, and go for the standard, today.

But why does BPEL matter? Out of the 250 BPM vendors one can find on Google, less than 10 support BPEL natively. The other 240 usually make a case that BPEL does not really matter, for no business analyst would use it directly. They're both right and wrong.

They are right in the sense that indeed, no business analyst—or **process analyst** for that matter—should ever have to write a single line of BPEL code. Instead, they will use a process modeling tool that will generate the code for them. But they are also wrong, for the code that is generated that way really does matter, for three main reasons.

First, using a process runtime that natively supports BPEL ensures that processes that are deployed onto it could be deployed onto an other runtime as well. As far as I can tell, this is the best insurance policy any customer could get. Admittedly, the fact that BPEL has multiple versions (1.0, 1.1, 2.0) and supports the definition of proprietary extensions—much like SQL with SQL-87, SQL-92 and proprietary stored procedures—makes such portability harder than it should be. Nevertheless, it's a lot easier than having to go from the proprietary language of Vendor Foo to the one of Vendor Bar.

Second, designing processes that are aimed at being deployed on a BPEL runtime allows developers to make some assumptions regarding the execution environment that will host them: web service interfaces will be there to interact with processes at any point in their execution, audit trail data will have a certain format—or at least be defined against a common semantic set, and the collection of process constructs that such processes are defined with is known in advance, without any ambiguity. This makes the development of standards-based tools for business activity monitoring and process analysis and simulation much easier within the BPM ecosystem.

Third, and whether workflow vendors like it or not—usually they don't, BPEL has become the de-facto standard in the industry. Most BPM vendors today used to be workflow vendors in the past, and the workflow community at large did a pretty good job of fragmenting the market, by making sure that nobody would ever develop nor support any useful standard. Think of it as a divide and conquer strategy where nobody actually conquers anything more than what was there at the first place. As a result, workflow vendors have a vested interest in downplaying BPEL, in order to slow down its adoption and their own evolution toward irrelevance. But make no mistake. When all major IT vendors—including IBM, Microsoft, Oracle and SAP—agree on any given standard, not much room is

left for others. QL might have been better than SQL twenty years ago, but IBM's support for SQL made it a sure winner. Support for XPDL would be a noble pursuit—albeit quixotic—if only XPDL was technically superior to BPEL. It's not, therefore I see no reason not to use BPEL. And if you happen to use a product that does not support BPEL natively, make sure to ask a BPEL vendor to help you migrate to one that does. Last time I checked, it was not all that difficult.

Last but not least, one of the best places to learn more about BPEL is on the blog of one of my top competitors, the excellent [BPEL Radio](#) published by Edwin Khodabakchian, Vice President of Product Development at Oracle. Edwin was the founder of Collaxa, the company that developed the first working implementation of BPEL. He knows what he is talking about, trust me on this.

 [Original Post](#)

BPMN

vs. ARIS, HIM, UML, Proprietary Notations

As for the process execution language, early BPM products sported many different notations, with cute little shapes and fancy colors. Some were very workflow centric, like HIM, others more technically oriented, like UML Activity Diagrams, but most were totally proprietary, incomplete, and incompatible with each other. BPMI.org set out to fix this, but this time around learned from its early mistakes and made sure that IBM was involved as early in the development process as possible. A fine gentleman by the name of Stephen White did his magic and developed BPMN, which quickly established itself as the standard notation for modeling executable business processes. BPMN supports both the orchestration of web service and the execution of human workflow tasks, while enabling the choreography of multiple business processes through the swimlane metaphor. BPM 2.0 works because one can go from BPMN to BPEL without having to write code. BPMN is not perfect and should learn a couple of tricks from HIM, but its support for compensating transactions, unsolicited events, complex loops and multiple swimlanes is what makes it unique, effective and irreplaceable.

But why does BPMN matter? Until now, there has been no standard notation for designing business processes. ARIS—the Brainchild of Dr. August-Wilhelm Scheer—is a great notation, but it's a very proprietary one that is not supported by any other tool than [IDS Scheer ARIS](#). [UML Activity Diagrams](#) are cute, but business analysts somehow cannot use them. [Flowcharts](#) is what they'd rather go for, but these tend to be limited to the modeling of single processes, which does not accommodate the requirements of a Service Oriented Architecture (SOA). Something better was needed, and this is why BPMI.org developed [BPMN](#) a couple of years ago.

Unlike BPML, BPMN immediately received the support of industry heavyweights such as IBM, which made it much easier to establish it as a standard. Also, unlike BPML, there was no real competition for BPMN.

To be fair, BPMN is not perfect yet, and version 2.0 is adding very little to version 1.0. A standard serialization format is needed (XML is not enough, being way too low level), and the way one can go from BPMN to BPEL is not fully specified. Adding to the complexity, nobody really knows how to go from BPEL back to BPMN, for there is no single way of doing this. Such a problem is also known as BPMN-BPEL round-tripping, and my friend Bruce Silver did a [good job](#) at describing it, following a [great article](#) from John Deeb and Devesh Sharma published by Business Integration Journal. Conventions will have to be defined, and I would be willing to bet that a standard round-tripping path will be defined sometime in 2007. In the meantime, vendors that support both BPMN and BPEL will have to give it their best shot. Too bad there are so few of them working on the problem today...

Eventhough BPMN still needs more work, it has the merit to exist, and nothing else comes even close. It's the first notation I have seen that can be used by business analysts, [process analysts](#), and software engineers alike, without much ambiguity getting in the way of their collaborative efforts. It's also one of these very rare notations that can provide a clear path to execution, something that is an absolute must-have if one needs BPM to do more than helping in the design of pretty pictures. As such, BPMN really matters and is one of the most powerful enablers for BPM 2.0.

 [Original Post](#)

BPMN Designer

vs. BPEL Editor

Early BPM products supporting the BPEL specification offered a BPEL editor as primary development tool, instead of a real business process design tool. The problem with this approach is that BPEL is a very complex specification, especially from a data management standpoint.

Furthermore, BPEL's heavy reliance on complex web services specifications requires developers to manually synchronize multiple BPEL and WSDL files in order to deploy an end-to-end process. BPEL editors do not make this exercise much simpler. They also produce process models that do not have a coarse-enough granularity for them to be shown to a business audience.

A higher-level notation is needed, it's called BPMN, and BPM 2.0 must take advantage of it for a wide-enough audience of process analysts to get the productivity they need for their BPM projects.

But what is a BPMN designer? Most BPM vendors that have developed native BPEL runtime components offer a BPEL editor as a development tool. This is better than having to write BPEL code manually, but don't expect business analysts—or even **process analysts**—to be productive with such a tool. BPEL is a very sophisticated process execution language, and as such was designed for computers, not human beings.

BPEL is a powerful language, but with power comes responsibility, or in this particular case, complexity. I won't bore you with too many technical details, but if you want to convince yourself that BPEL is not for the faint of heart, just try to write the chunk of BPEL code you would need to correlate two process instances, say one for an order taking process and the other for an order fulfillment process. If you do, you'll quickly realize that you'd rather have a tool do it for you when you draw an arrow going from one box to another. That's precisely what a BPMN designer does, alongside many other similar tricks.

From a distance, a BPMN designer seems awfully similar to a BPEL editor: they both look like glorified Visio-like diagramming tool, they both show neat little boxes and arrow, they both let you change colors and fonts, but the comparison pretty much stops there. Where the first only requires that you get a good understanding of the different types of flows that BPMN supports, the second mandates that you know the ins and outs of the BPEL specification. Not only that, but you'll also have to master the WSDL specification if you want to build a process that does anything useful, and that's where things get really tricky. WSDL might have one of the smallest specifications of all the different standards for Web Services, but it's also one that will require at least 10 reading sessions before you being to remotely understand what its authors had in mind when they wrote it. To make a long story short, I love BPEL, but I love it even more if I do not have to deal with it directly.

To really grasp the difference, you do not have to go very far. Instead, just read carefully. On one hand, a BPEL editor is exactly what it says it is: an editor for BPEL code. As such, it's aimed at people who can read and write BPEL code themselves, and such people tend to be very sophisticated software engineers. They know Java, most of the J2EE APIs, and any specification which name starts with 'WS'. On the other hand, a BPMN designer is for the rest of us. It's a tool that will generate BPEL code out of a BPMN designer—if you picked a good one that is, but won't necessarily let you mess with the BPEL code itself, because roundtrip engineering between BPMN and BPEL is a really tricky problem to solve. [eClarus](#) is about to release a version of their tool that does it, but I'll reserve my judgement until I see customers managing to go back and forth between the BPMN diagram and the BPEL code. And beside, BPM 2.0 should support [Zero Code](#).

So next time you look for a business process modeling tool that will generate code for you, look beyond the pretty pictures. If the tool requires that you master the BPEL specification, make sure that you do before committing. And if the tool generates anything else than BPEL, make sure that you read this [article](#) first. If you do both and happen to fit the profile of a process analyst, you'll come to the same conclusion as I did: a BPMN designer is the way to go.

 [Original Post](#)

Zero Code

vs. Writing Code Behind the Boxes

BPMN and BPEL make for an extremely powerful combination because they allow one to go from picture to code without having to actually write the code. Let's face it, a lot of work went into the development of these two specifications, and this work benefited from an unprecedented amount of collective experience that no single vendor could ever match on its own. What that means is that most BPM products that are based on proprietary notations and execution languages actually require the writing of quite a bit of code in order to make processes executable. Double click on the neat-looking boxes and arrows, and code written in Java or proprietary languages will show its face. There is nothing fundamentally wrong about code, but it just so happens that writing and maintaining code is harder and more expensive than writing and maintaining none at all. BPM 2.0 makes it possible to implement the most complex processes without having to write code. The Dutch Government did just that for a process that has a quarter of a million activities, so if it worked for them at such a scale, it should work for many other organizations.

But why does Zero Code matter? This is the eighth edition of our weekly BPM 2.0 post. Today, I will try to explain why **Zero Code** matters. Most BPM solutions require users to manually write code behind boxes and arrows used to depict a process. There is nothing wrong about software code, but code is one of these things for which less is more, and there is more than one reason for it to be true.

First, BPM needs the participation of business analysts, but business analysts cannot read nor write code. Granted, technical people could write the code behind the boxes and arrows that business analysts would draw, but it would not really help bridge the business-IT divide, would it? BPM 2.0 advocates a model where executable processes are implemented by **process analysts**, and these could write code if they really have to, but they tend to prefer using graphical tools instead.

There are 8 million Visual Basic programmers today—it's the largest developer community ever built—and there must be a reason why Microsoft called its BASIC development tool Visual. If no business logic can hide behind the boxes, business analysts are more likely to understand what process analysts have done, and this is a sure way to bridge the business-IT divide, if not obliterate it entirely.

Second, human beings make mistakes, and the exercise of writing code is an error-prone one. In average, one BPMN process shape leads to 10 lines of BPEL code, and one line of BPEL code replaces approximately 10 lines of J2EE code. What this means is that 100 lines of nasty J2EE code would have to be written if BPEL generation tools did not exist. If your process has hundreds or thousands of steps (or 250,000 like the one [Intalio](#) built for the Dutch Government), that's tenths of thousands or hundredths of thousands of lines of code. Commercial code typically has anywhere from one to seven bugs per 1000 lines of code according to a report from the [National Cyber Security Partnership's](#) Working Group on the Software Lifecycle (Source: [ZDNet](#)).

What this means is that your average BPM project will have hundredths, if not thousands of bugs in it, which will have to be fixed manually, one by one. If you let a Zero Code BPM tool generate the code for you, you do not have to worry about these bugs anymore. Granted, the BPM tool's code generator could have some bugs, but it's likely that they'll be fixed by the vendor before you could catch the bugs you'll create yourself by writing code manually. Furthermore, once the tool's bugs are fixed, they produce valid code anyone using it.

Finally, software code is usually imperative, while Zero Code design-driven development tends to be much more declarative in nature. What this means is that information contained within BPMN diagrams is much more explicit than equivalent information embedded into J2EE or C# code. The more explicit the information, the easier it gets for the BPM platform to provide simulation, debugging, reporting and optimization capabilities. To make a long story short, code is better written by the vendor and customers should focus their efforts on improving their business processes rather than code is used to support them.

 [Original Post](#)

One Click Deploy

vs. Writing Deployment Descriptor Files

By their very nature, business processes are prone to change, and most of us got interested by this new BPM thingy because of the promise that it would make change a little bit easier. I can even remember one of the early pure play BPM startups using the tagline “Go ahead! Change!” to emphasize that very point. Well, this is all nice and fancy, but if one has to write multiple deployment descriptor files and configure various web service interfaces to deploy a process, the ability to change the process as you go remains a pipe dream. BPM 2.0 advocates a radical ‘One-Click-Deploy’ approach to solve this problem. Once your process is valid, with all data mappings completed, business rules defined, and workflow parameters set, just click on a button and get the process deployed on your runtime environment, without any additional work. There is absolutely no reason why it should be any more complex than that.

But why does One Click Deploy matter? This is the ninth edition of our weekly BPM 2.0 post. Today, I will try to explain why **One Click Deploy** matters. An executable business process is more than a pretty picture. To start, it might be made of multiple such pictures, which must be kept in sync. But behind the neat little boxes and arrows lie very many artifacts that need to be taken care of as well. If one is to make frequent changes to a business process—whatever the reasons for this might be—the underlying Business Process Management System (BPMS) has to make it very easy for such changes to be reflected at the execution level, and One Click Deploy is what makes it possible.

At a very high level—and assuming a BPEL deployment model—a process is made of flows, services, data transformations, business rules, workflow task definitions, and user interfaces. From a runtime standpoint, these lead to the generation of multiple files. Flows, services and transformations are expressed in BPEL, WSDL, and XPath or XSLT respectively.

There is no standard for business rules yet, therefore they have to be expressed in a proprietary format for the time being. For workflow tasks, the **BPEL4People** model can be used, but no formal specification exists for it either, therefore proprietary deployment descriptors are needed. And as far as user interfaces are concerned, there are more options than I have fingers and toes on my hands and feet—and I have been lucky to keep all of them so far, therefore whatever goes there will be highly vendor-dependent. Intalio opted for the **XForms** standard, but I would not claim that it should be a requirement for **BPM 2.0**. At the very best, it's a good candidate.

Problem is, with so many files, the deployment of a single process can quickly turn into a nightmare if each and every file needs to be manually deployed. And if you need to compile them, or provide specific deployment descriptor for them, the task becomes almost impossible for most. This is where One Click Deploy comes in handy. The idea is pretty simple: use a process design tool that supports a **Zero Code** development methodology and get all process artifacts automatically deployed on their respective runtimes in a single mouse click.

When we started talking about One Click Deploy, some experienced IT managers objected that it would break best practices that call for the use of a development server and a staging server before moving to a production server. They should not have been alarmed, for One Click Deploy also works that way, in the sense that you might need three steps to take a process from design to production, but each step should take no more than one click: one to push your design to a development runtime environment, an other to migrate it to a staging environment, and yet an other to upgrade your working set of artifacts to a real production system.

If that sounds too simple to you, it should not come as a surprise, for a lot of engineering work goes into the development of a product that can support such a streamlined life cycle for executable processes developed in a Zero Code fashion. In other words, there is no magic pixie dust, just plain old blood, sweat and tears that software engineers developing BPM 2.0 products have shed so that you do not have to.

 [Original Post](#)

Generating Web Services on-the-fly

vs. Implementing Application Connectors

Most BPM solutions are workflow systems in disguise, and as such, they do not really support integration with back office systems. Distributed transactions and reliable messaging are foreign concepts for such tools. For the rest of them, integration with enterprise applications has been forever tainted by what could be considered as the biggest scam in the history of enterprise software, the idea that you need custom connectors to integrate with enterprise applications such as PeopleSoft or SAP. In the late nineties, EAI vendors made a fortune selling connectors: you want to enter a purchase order into this version of SAP R/3? Buy connector X, for a cool \$25,000 per CPU. You want to get the list of employees from that version of SAP R/3? Buy connector Y, for an other \$25,000 per CPU. In reality, one can write a generic connector for all versions of SAP R/3, back to SAP R/3 3.1i, that exposes all 200,000 BAPIs, IDOCs and RFCs as web services, on-the-fly, for both standard and custom transactions, without writing a single line of code. The same can be done for Oracle, PeopleSoft, Siebel, and most enterprise applications out there. If it's possible, BPM 2.0 should take advantage of it, and unless one takes some masochistic pleasure in developing one-time connectors for APIs that might be obsolete the next day, nobody should have to write custom connectors anymore.

But how does BPM 2.0 relate to SOA? Unless you're still living in the workflow-centric world of the 90's, you know by now **why BPEL matters**. Problem is, the only thing BPEL understands is web services, and only one very narrow type of web service at that—WSDL. Here is the bad news: if you need to orchestrate transactions that are not yet exposed as web services, BPEL won't help you. BEA suggested support for Java with the **BPELJ** specification, but I do not know any **process analyst** who likes to write Java code, so we'll pass, thank you very much. Now the good news: a good BPM 2.0 product can give you web services for free out of pretty much anything out there.

According to the BPM 2.0 model, web service interfaces should be generated on the fly for any application or middleware system that supports some form of API, and there are very few that do not today. Even legacy mainframe systems can expose APIs through the use of common screen-scraping tools. A good BPMS will be able to leverage these APIs and generate web service interfaces for it. Furthermore, the BPM 2.0 model also states that such interfaces should be offered for both inbound and outbound transactions, meaning that a BPMS can invoke a transaction with a third-party system through a web service interface (inbound), while a third-party system can call a process deployed on a BPMS through a very similar interface (outbound).

This answers a question that we get all the time: “do I need SOA to use BPM?” The answer is both “yes” and “no”. Yes, you need SOA to use BPM, for that’s how the BPEL execution model was designed. But no, you do not need to have your Service Oriented Architecture in place before being able to use BPM, for the mere installation of a good BPMS will SOA-enable most of what you already have, or at least it should. If you pick the right BPMS, it will come with an Enterprise Service Bus (ESB) and a Service Repository that are the cornerstones of any Service Oriented Architecture worth using. In other words, buy BPM 2.0 and you’ll get SOA for free. And if you’re lucky enough to pick an Open Source BPMS, you won’t even have to pay for it. Is this a good deal, or what?

 [Original Post](#)

Interpreting BPEL Code Natively

vs. Generating Java Code

Early implementations of the BPEL and BPML specifications relied on Java code generation: you write the code in BPEL, and a code generator automatically translate it into a set of Java classes that are deployed on a Java Virtual Machine or a J2EE Application Server. Good news: it's a relatively easy way for a software vendors to get into the BPEL game. Bad news: it does not really work. Much like Oracle's database does not generate C code to execute a given SQL query, a good BPMS should not have to generate Java code to execute a BPEL process.

Java code generation is bad because it makes the deployment of processes more complex than it should be, it creates discontinuity from the process semantic that makes debugging and monitoring an order of magnitude more difficult than with native interpretation, and ultimately, it slows everything down.

Things get even worse when such implementations rely on the EJB component model for the persistence of process data, especially when Entity Beans with Container-Managed Persistence are being used. For it to work at a large scale—the aforementioned Dutch Government is running 250 million concurrent process instances that take up to five years to complete on a 4-CPU box—BPM 2.0 must natively interpret the BPEL 2.0 code, ideally through just-in-time compilation into process bytecode that closely maps to the Pi-Calculus semantic, and forgo the use of any EJB component for persistence, relying instead on straight database connectivity. If you need performance and scalability, you should go for such a model.

But why does native BPEL interpretation matter? When you are a software vendor and you have invested tens if not hundreds of man-years into the development of some proprietary process engine, it is very tempting to add a simple translation layer that would turn BPEL code into what your engine can digest. Customers should shy away from such solutions though, for they create more problems than they really solve.

Some of the issues to be aware of are described in the [original article on BPM 2.0](#), therefore I will focus today on some additional considerations. First and foremost, translating BPEL into another proprietary language creates the temptation to modify executable code at that level, which adds a third layer underneath BPMN and BPEL, and makes the task of supporting roundtrip engineering twice as difficult as it should be. To keep things simple, BPMN should be serialized in a canonical way using technologies such as the [MOF](#) or [EMF](#), then translated into BPEL code at deployment time. This is pretty much how client-server database development tools have been working using UML and SQL for years, and process development tools are not much different in that respect.

Furthermore, not adopting a native BPEL execution model creates an impedance mismatch between the semantics of process execution defined by BPEL and the one supported by the proprietary runtime. In most cases, this leads to the development of process engines that will only support a subset of the BPEL specification, while adding some extra features that require proprietary extensions. If you buy into industry standards, this is bad, plain and simple.

Additionally, vendors that invented their own language and built support for it within their process design tool and their process runtime usually have a base of legacy customers to take care of. If the migration strategy is to support both languages within both the tool and the runtime, the required engineering effort is at least four times as significant as supporting a single execution language. As a result, it is very likely that the needs of legacy customers will prevail over the needs of prospective customers, and support for BPEL will be poor at best.

Like it or not, BPEL is a very powerful yet extremely complex language, and supporting it properly is no simple task. If you're a BPM vendor and find yourself in the position where customers are asking for it, I would strongly recommend that you embrace it fully and migrate your existing customers to a next-generation product that implements native support for the specification, and nothing more. And if you're an end-user, my advice is even more radical: do not consider using any product that is not natively built around BPEL, for you will pay the price of not being standard-compliant down the road.

 [Original Post](#)

Web 2.0 User Interface

vs. Web 1.0 User Interface

BPM 1.0 solutions relied on Web 1.0 user interfaces, namely standard email clients and web portals serving task lists and forms produced using plain HTML. BPM 2.0 must take advantage of Web 2.0 and Office 2.0 technologies, such as AJAX for dynamic tasks lists and complex forms supporting client-side data validation, RSS feeds for process events and user task lists, weblogs and wikis for process documentation, web-based calendars for task scheduling, and REST APIs for supporting the most creative mashups. Somehow, BPM has yet to be perceived as a ‘cool’ technology, and Web 2.0 might be all that is needed to move the needle enough to get a more mainstream audience excited by it.

But why does a Web 2.0 user interface matter? Like with any other application, the most difficult part in deploying a business process powered by a BPMS of sorts is in getting active support from end users. For the deployment to succeed, the application has to be actually used by end users, and user interfaces play the most critical role in this.

The problem with user interfaces is that they are not what they look like. A user interface could look ugly and still be a good one, from the standpoint that end-users would like it. End users like user interfaces not because they look sexy, but because it helps them get their job done, better than when using anything else. And when it comes to user productivity, the user interface’s workflow has a lot more impact than the color of its icons.

User interface workflow covers things such as the requirement to install a client application or not, the number of steps required for such an installation, the way end-users get notified when they have to use the user interface in order to achieve a certain task, and the number of steps they have to go through in order to get more information about the task and to get it done.

User interfaces built with a traditional client-server architecture required the installation of dedicated clients for example. This problem was solved with the emergence of web-based user interfaces, but the use of static HTML significantly increased the number of steps—usually quantified in mouse clicks—a user would have to go through in order to complete certain tasks, sometimes by an order of magnitude or more.

Next-generation user interfaces powered by AJAX technologies and leveraging RSS -type feeds bring together the best of both worlds into a significantly-improved end-user experience. When combined with the right set of server-side process automation technology, they can accommodate both push and pull notification mechanisms, provide dynamic user interfaces with responsive data lookup and drill-down capabilities, and support the complex sequencing of activities that will maximize end-user productivity by removing the need for logging to multiple applications or dealing with multiple individual tasks.

Down the road, [Office 2.0](#) should become the primary user interface for [BPM 2.0](#).

 [Original Post](#)

Rule Engine Included

vs. Bring your own Rule Engine

Up until now, BPM solutions would fall into two camps: you either had a glorified rule engine presented as a generic BPM solution, or you had a generic BPM solution that failed to support the execution of complex business rules natively. As a result, most customers who deployed a BPM solution of the later kind had to look for a rule engine from a third-party vendor, even though they did not really need a full-fledged rule engine to bring with. BPM 2.0 makes the rule engine a requirement, so that it can be leveraged by the BPM vendor itself in places where it makes sense, such as decision branching, message routing, late-stage service binding, or contextual user interfaces. As James Taylor puts it, it is no longer OK for the BPM vendors to have nothing in the way of a rule engine—they must either build something comparable or, more likely, OEM something from one of the business rules leaders. BPM 2.0 makes the Business Rule Management System (BRMS) part of the BPMS, so that only one platform has to be managed and the lifecycle of rule-driven processes can be streamlined. To a large extent, the BPMS becomes the killer application that rule engine vendors have been waiting for.

But why does a BPMS need a business rule engine? Unless your business processes are extremely complex, you might never need the power of a full-fledged business rules engine, but this does not mean that your BPMS should not include one by default. There are many reasons for this.

First, there is no standard way of integrating with a business rules engine, and if there ever was one, there would be no standard way of describing a business rule like there is for business processes using BPMN serialized into BPEL code. Vendors of business rules engines have tried to build such a standard for years, but they never could agree with each other, and I strongly doubt that they ever will. As a result, if a BPMS does not include a business rules engine by default, it has no way of providing a standard interface that would allow customers to plug their engine of choice.

Second, the only way to really take advantage of a business rules engine within the context of a BPMS is to align the lifecycle of business rules with the lifecycle of business processes. For example, a business rule will have to be expressed against the data model of objects or services that are orchestrated by business processes. When the process changes, the data model of its related objects and services will change as well, which will impact business rules using them. If the two lifecycles are not aligned, you can forget about [One Click Deploy](#).

Third, the BPMS should take advantage of the capabilities offered by the business rules engine not just for business rules, but also late-stage binding rules to select which implementation of an abstract service should be used, how a user interface should be tailored to specific end-users, and when to escalate an alert to an IT manager when some system-level exceptions are generated.

If your BPMS of choice does not include a business rules engine by default, you will still be able to invoke business rules through some kind of service interface or API, but you will lose the benefits outlined above. Interestingly enough, this is one of the pieces that are missing in the Intalio|BPMS, and we are aggressively working to fix this. Our two main options are [Corticon](#) and [Drools](#). The first is nice but not free. The second is Open Source but does not have an Eclipse-compatible tool yet (at least not a full-featured one). Shall we go with the second option, we will develop a tool through our [Demand Driven Development](#) program. If you have any interest in this, feel free to [drop me a line](#).

 [Original Post](#)

Real-Time BAM Included

vs. Bring your own BAM

Much like the data management industry had separate vendors for data processing (database vendors) and data analytics (business intelligence vendors), the business process management industry has featured separate vendors for BPM and BAM. It might take time for companies to actually merge, but products won't wait for this to happen before merging on their own. With BPM 2.0, BAM is part of the overall solution, day one, instead of being an optional afterthought. You need BAM, because doing BPM without it is like driving a car with your eyes wide shut. BAM is one of those godsend that BPM makes possible, and I cannot think of a reason why anyone should not take advantage of it today.

But why does a BPMS need a BAM platform? In a nutshell, the reasons are the same as why a BPMS should include a Business Rules Engine and which were heavily debated following last week's [article](#).

First, there is no standard way of integrating a BPMS with an off-the-shelf Business Activity Monitoring (BAM) platform. If you think that a standard event interface will do the trick, you should think twice. The reason for this is pretty simple: aggregating events without the map of the process that is generating them at the first place is like re-inventing the wheel everytime you put a new process into production. And because there is no standard format for exchanging process models—BPMN is too little, BPEL is too much, and XPDL is off-base—your best bet is to go with a BPMS that supports BAM out of the box.

Second, much like the lifecycle of business rules must be synchronized—read my lips: synchronized, not aligned, syn-chro-nized—with the lifecycle of business rules, the lifecycle of business metrics must be synchronized with the lifecycle of business processes they relate to. Old-fashioned workflow metrics could not care less about process data and dynamic schemas, but this is not true anymore with BPM, or at least not with BPM 2.0. Having the BAM infrastructure part of the overall BPMS platform addresses this lifecycle issue.

Third, a BAM tool is actually the best helper you can get for process debugging and simulation at design time. Having it tightly integrated with your process design environment is a good way to achieve [Zero Code](#) and [One Click Deploy](#) for end-to-end processes, throughout their entire lifecycle.

And much like [Intalio |BPMS](#) does not include a business rules engine yet, it does not provide out-of-the-box integration with the BAM platform we decided to use moving forward—[Celequest](#). This is something that will have to be developed through our [Demand Driven Development](#) process, and I invite you to join our [developer's community](#) to learn more about it.

 [Original Post](#)

Native Process Simulation

vs. Ad hoc Process Simulation

With BPM solutions of the first generation, process simulation was supported by ad hoc process simulators that were based on primitive finite state machine simulating the execution of processes to be deployed on a separate runtime. Good news: such an ad hoc simulator is relatively straightforward to implement. Bad news: it does not reflect the true nature of the target runtime environment, cannot accurately simulate load testing, and has no visibility onto process data, which represents a good half of the process' semantics when using process execution languages such as BPEL. While appropriate for addressing the needs of business analysts who have no interest in the actual execution of the processes they model, such simulators are totally useless to the people who own the overall process lifecycle.

Learning from this experience, BPM 2.0 advocates a different approach for simulation, whereby the process engine is used as process simulator. According to such an approach, simulated processes are stubbed out from external systems, which are emulated by the process engine itself. In order to simulate a process, the process development environment automatically deploys a collection of process instances and randomly generate seed variables in order to support simulation models such as Monte Carlo. The process engine executes the set of simulated process instances and lets the BAM infrastructure aggregate the results to be displayed back to the process analyst, who gets access to business-level key performance indicators, as well as system level performance metrics.

This approach ensures that the semantics of the simulated process remains 100% accurate with respect to the semantics of the process to be deployed. Also, by combining business indicators with system metrics, it guarantees that business and IT get equal representation in the evaluation of processes to be deployed within a mission-critical production environment.

At the end of the day, ad hoc process simulators are nothing more than cute toys for business analysts, while native process simulators are accurate metrology instruments that BPM practitioners can safely rely on.

But why should a BPMS support process simulation? The [original BPM 2.0 post](#) suggested that the core process engine should be used as process simulator. Yet, it did not provide much explanations as to why a process simulator is needed at the first place. From an IT standpoint, business processes can be viewed as long-running transactions. An order-to-cash process might take weeks to complete, while instances of a hire-to-retire process could run for many years. As a result, simulating the execution of business processes requires the ability to fast-forward in time and playback process fragments, while making modifications to process models until simulated process instances deliver appropriate results. In a nutshell, fast-forward and playback are what a process simulator does.

In so doing, a process simulator must also emulate the behavior of external systems that cannot be involved at simulation time, such as ERP systems for example. This explains why some people prefer to call it a process emulator. When feeding a process model to a process simulator, the later emulates third-party systems by stubbing them out and providing simple ways for the [process analyst](#) to specify default inputs and outputs for stubbed-out remote transactions. This emulation interface can also include randomly-generated parameters used to implement simulation methods such as [Monte Carlo](#)'s.

When using the core process engine as process virtual machine for the process simulator and the [built-in Business Activity Monitoring platform](#) as user interface, the same infrastructure can be used by both business types and IT folks. The former will simulate the process at a high level while underlying transactions have been stubbed out by process analysts, and the later will simulate the process at a low level while paying special attention to system-level metrics such as transaction latencies and caching parameters. When used by a software architect, a process simulator becomes a fabulous process testing and debugging environment. Of course, different interfaces have to be exposed to both categories of users, but the underlying infrastructure can be the same.

 [Original Post](#)

Dynamic Process Optimization

vs. Continuous Process Improvement

Adepts of the Business Process Reengineering school promoted the concept of continuous process improvement, and early BPM vendors made sure to support this methodology with their products.

Problem was, changing the process once deployed into production turned out to be more difficult than most had initially expected, especially when software code had to be re-written for changes to be applied.

Furthermore, if making a change to a process meant going through the entire process lifecycle all over again, from modeling to simulation and deployment, most ideas for process improvements remained just that, ideas.

BPM 2.0 marks a departure from the concept of continuous process improvement, and promotes a more dynamic process optimization model, whereby key process elements can be optimized on the fly, without having to re-deploy the entire process. Through the use of native BPEL interpretation, reusable process interfaces, externalized business rules, late-stage binding, and instance-level exception handling, running process instances can be optimized in real time, without requiring advanced technical skills.

Such an approach allows the Dutch government to make regular changes to processes that can take up to five years to complete. If anyone had any doubt that BPM could be used to support long-running transactions, such doubts should be put to rest by now.

But what is dynamic process optimization? A couple of years ago, the tagline for a now-defunct BPM company was “Go ahead. Change.”, and the ability to make rapid changes to processes has always been heralded by vendors as one of the major benefits offered by BPM. Dynamic process optimization is all about taking this idea a step further.

One could define agility as being the speed at which changes can be applied to business processes that have been deployed into production. According to such a definition, things like [Zero Code](#) and [One Click Deploy](#) should play a critical role in fostering agility within an organization. Nevertheless, no matter how easy we make it to go from design to testing, then from testing to staging, and then from staging to deployment, the BPM process lifecycle becomes an agility impediment when real-time changes are required.

In order to work around such limitations, dimensions along which processes are the most susceptible to change must be extracted from the core process lifecycle. Parameters and variables that drive process execution must be externalized, and the lifecycle of such variables must be either totally decoupled from the lifecycle of the business process they relate to, or at least merely synchronized with it.

This is true for business rules—as was discussed at length in [comments](#) to this past article, but it is also true for binding rules invoked to bind abstract process and service interfaces to their matching implementations, as well as user interface parameters used for internationalization, localization and personalization purposes. In order to support real-time changes to be applied to business processes, it should be possible to change such parameters and variables at runtime, without having to redeploy the processes themselves.

The challenge then becomes about how to provide simple-enough user interfaces that can let business users make such changes at runtime, while enforcing strict privacy and security rules, and capturing a reliable audit trail for such interventions along the way.

Down the road, the most forward-thinking developers will want to close the loop and design processes that will change such parameters on their own, based on information gathered at runtime by the BAM infrastructure, and without any human interventions. To some, it will sound like science fiction. To others, it will look very similar to what genetic algorithms have been used for in many mission-critical production systems for the past ten to fifteen years.

 [Original Post](#)

Open Source Process Engine

vs. Closed Source Process Engine

Your process engine will quickly become the most critical piece of your IT infrastructure. As such, you need the best insurance policy money can buy for it, and if you can get it for free, even better. This is the main reason why your BPMS should be architected around an Open Source process engine. Whatever should happen to your BPM vendor of choice, a community will remain to support you. The web, which one could have called Internet 2.0, was literally built on top of the Open Source Apache web server, which still runs a cool 67% of all web servers connected to the Internet today. The same will be true for BPM 2.0, and the leading Open Source BPEL server will also become the leading BPEL server.

But what is the rationale for an open source process engine? Beside the fact that an Open Source license provides the best insurance policy a customer of enterprise software can get, the Open Source development process brings many benefits that are particularly relevant to a process engine.

First, it takes very significant resources to develop any piece of infrastructure software to be used in a mission-critical environment, be it an operating system (think Linux), a database (think MySQL), a web server (think Apache), or an application server (think JBoss). The Open Source development process has been very effective at producing such software in a reliable fashion. Because the software gets deployed by more users, and because the source code is accessible to more contributors, Open Source offerings tend to outperform any closed source alternative over time. This is true for a process engine as well.

Second, the semantic of process execution at runtime is critical to some applications, especially for compliance requirements or security reasons. When the code of the process engine is open for all to see, no ambiguity can exist as to how processes will actually execute. And if the engine's developers made an error in interpreting the BPEL specification, anyone is free to contribute a fix that will benefit the entire community of users.

Third, processes are everywhere, yet Workflow and BPM technologies have failed to gain mainstream acceptance, even though they've been available for more than 15 years now. One reason for this is that until now, BPM products have been built upon proprietary process execution models. Also, they have proven to be quite difficult and expensive for Independent Software Vendors (ISV) to embed within their own products. The wide adoption of BPEL is fixing the first problem, while the availability of a rock-solid Open Source BPEL engine is expected to address the second.

For all these reasons, we need a good Open Source BPEL engine.

 [Original Post](#)

Get Started Today, Free of Charge

vs. \$250,000 Entry Fee

Last but not least: BPM is too good of a technology for it to be kept out of reach from most of its potential users because of high acquisition costs. Until now, a fully featured enterprise-class BPMS would cost north of \$250,000. If you wanted to add support for business rules, BAM and a couple of enterprise applications, it would cost you close to \$500,000. I tend to think that such a high price tag is the largest single contributor to the slow adoption rate for BPM that we witnessed over the past six years.

BPM 2.0 will change this by making enterprise-class BPMS products free of charge when deployed through certain configurations. Hybrid business models blending Open Source and commercial software will ensure that a handful of vendors enjoy the success they need for supporting customers over the long run. In the end, customers, system integrators and software vendors alike will all benefit from the upcoming explosion of the BPM market. Things are starting to get exciting, so don't wait any longer and join the party today!

But why do we need a free BPMS? I believe everybody will agree that having something for free is always nice, and as my good friend Bruce Silver once said, **why buy a BPMS when you can get one for nothing?** Actually, our competitors might disagree, but I can live with that. Instead, I will tell you a little bit more about what's coming next on the BPM 2.0 front.

First, I'll package the **original BPM 2.0 article** and subsequent 18 weekly posts into a single white paper, following the advice given to me by Steinar Carlsen. Soft copies will be freely downloadable from the IT|Redux website, while hard copies will be available for purchase online. I have tried to use Office 2.0 applications for publishing this document, but the lack of proper support for pagination made the exercise virtually impossible. I guess I will have to use a regular word processor for the time being.

Second, I must admit that I lied in the introduction to this post. This is not the last edition of our weekly BPM 2.0 post. Next week, we will start a new cycle, going through the same 18 principles for BPM 2.0, but this time around talking about how actual customers are getting benefits from the application of these principles. I have no idea whether I will be able to come up with actual customer examples for all 18 of them, but I will do my best. I will also try to relate to [Intalio|BPMS](#) when applicable, but you will soon realize that as much as I love this product, it does not yet comply with all 18 principles. If you ever thought that BPM 2.0 was just a shameless attempt at promoting the company I work for, think again. I am promoting a vision, and my work at Intalio consists in trying to deliver on it, not the other way around.

Third, I will try to explain why BPM really is cool. Recently, there have been some discussions [here](#) and [there](#) about whether BPM is cool or not, and whether it should matter at the first place. Well, to me, BPM really is cool, much like a database or an operating system can be. If you cannot fathom such a thing, I won't hold it against you. After all, everybody is free to get a kick out of whatever they like. But as far as I am concerned, BPM, or what I call BPM 2.0, might be the most important development the enterprise software industry has seen over the last twenty years. On this front, the BPMS is on par with the RDBMS, and the lives of many a developer were changed with the introduction of such a technology. Not in the same way as the introduction of a new vaccine would, but in a very significant way nonetheless, and if that is not cool, I do not know what is. Now, to many, BPM remains an obscure thing, something they have a hard time relating to, and my job on these pages is to change this perception and to show that BPM is real, powerful, effective, and simply cool.

 [Original Post](#)

About the Author

Ismael Ghalimi is a passionate entrepreneur and fervent industry observer, founder and CEO of [Intalio](#), creator of [BPMI.org](#) and initiator of [Office 2.0](#). Ismael is a professional scuba diver, private pilot, and fanatic American V-Twin rider. Ismael can be reached at ismael@itredux.com.