Developer-Friendly Business Process Management

May 2014

Kemsley Design Ltd.

www.kemsleydesign.com
www.column2.com

Overview

Many BPM experts today will tell you that the key to business agility is business-led, zero-code, model-driven development. In other words, a non-technical analyst creates a graphical process model that generates an executable process application without writing code. In fact, for the past 15 years, vendors have attempted to make business process management systems (BPMS) more user-friendly and analyst-friendly through the use of model-driven design and development. These "zero-code" paradigms allow non-technical participants to create their own executable process models, typically using the Business Process Model & Notation (BPMN) standard.

Beyond simple, non-integrated processes, however, the reality is that most BPMS projects involve technical developers as well as non-technical analysts. Unfortunately, the proprietary model-driven environments that provide a simple interface to analysts can provide barriers to developer efficiency and code transparency.

This paper explores the myth of zero-code BPM in complex projects, and how proprietary design and development tools can hinder – rather than help – skilled enterprise developers. It considers how using standard development tools and coding languages in conjunction with a BPMS can provide a bridge between business and IT without forcing either to use tools that are inappropriate for their needs: business-led functional design paired with efficient development in corporate-standard application development environments.

Limitations of Zero-Code, Model-Driven BPM

Before the popularity of model-driven BPMS, the business wrote requirements and handed them off to IT, then IT went away for months – or years – and developed a business process application that integrated with other line of business systems. There was very little collaboration between business and IT, just a one-way flow of written requirements and change requests; the business rarely had the opportunity to collaborate during development, and the resulting system often bore little resemblance to what the business really needed.

Model-driven BPMS improved business agility and business-IT collaboration significantly: business analysts now create a graphical process model that can immediately generate an executable process without writing code. Prototypes can be completed in hours or days, then directly augmented by developers with additional technical functionality. Changes to the process flow can be made quickly in the graphical environment, and the BPMS provides a full application development environment not just for process flows, but for integrating services and rules, creating user interfaces, designing analytics, and much more.

It sounds idyllic, but there are some serious flaws with this paradigm: first, non-technical business people are unlikely to design and maintain their own executable processes with no help from IT; and second, developers are hindered by the proprietary nature of a full BPMS application development environment.

In reality, business users and analysts generally don't create complete executable processes in a BPMS unless the processes are fairly simple, the processes involve only human-facing steps, and the user interface for the user tasks within the process is automatically generated (or nearly so). Once the process requires more complex event-driven logic, or needs to integrate with services to perform certain functions, or must be integrated into a corporate portal, developers must write code to bridge from the original process model to an executable process-based application. The acceptance of BPMN as a standard process modeling language has removed a significant barrier to business-IT communication of process models, but it has also created a challenge: since the full BPMN specification is so rich, it can be a complete visual programming language that can be used to create models that are

no longer comprehensible to the business: effectively, graphical code. Often, a business analyst creates an initial process model, then a developer adds a significant number of elements to the model – as much as doubling the size and complexity of the model by adding tracking, notification, event-handling and automation steps – in order to make it executable. These additional elements are not relevant to the business view of the process, but create "noise" that makes it very difficult for the business to continue to understand, much less maintain, the process model that they originally created. Just as in the days prior to model-driven design, the business ends up creating a requirement in the form of a process model, then it is passed off and transformed by IT into a technical process model that is no longer understandable to the business. The main "user" of the model-driven BPMS becomes the developer rather than the business, contrary to the vendors' messages when these systems are sold to organizations.

Even if the process is not modeled using BPMN, any zero-code model-driven development will eventually become sufficiently complex (for all but relatively simple processes) that it is no longer understandable by business users, regardless of the fact that it is being created in a visual zero-code environment.

Being model-driven does not imply that no analytical or technical skills are required; that is more a function of the complexity of the model and application, not the application development tool.

This exposes the second flaw of zero-code BPM: when developers are tasked with transforming those initial process models into process applications, they are forced to use BPMN as their programming language and create the entire application within the constraints of the BPMS vendor's proprietary development environment: a "walled garden" that typically would not be considered best of breed as a pure application development tool. If we accept that the technical process models are not intended for business-IT communication, then why force skilled developers to use languages and tools that hinder their work? Why not make BPM more developer-friendly by using executable process models for the portions of the process that are relevant to the business, but allow developers to use more powerful languages such as Java to complete the non-business portions of the application, and standard corporate development tools for building, testing, integrating and maintaining these applications?

Drivers For Developer-Friendly BPM

A fully model-driven BPMS is a good solution for simpler processes, allowing less-technical participants to create executable processes and applications. However, for highly-skilled development teams creating high-performance, complex applications, the advantages of model-driven BPMS need to be combined with those of enterprise-strength application development environments.

Often this is an issue of corporate development standards and existing applications: large organizations have already invested in creating the optimal development frameworks and teams for their own needs, and have large application portfolios built on those standards. They are unlikely to refactor all of their enterprise applications onto the BPMS platform: BPM is a functionality that they want to incorporate into their existing applications using their existing tools, rather than use it to replace their existing framework. Compared to most enterprise application development tools, a model-driven BPMS falls far short for general application development (although it may be optimal for simpler process-centric applications), in part because it lacks the open, best-of-breed approach that allows any part of the stack to be swapped out.

One of the biggest challenges with full-stack BPMS products is that they can't easily take advantage of the standard components that are already in place in the enterprise, such as corporate portal user interfaces. Although they can make, or be invoked by, service calls to integrate with these other components, there is often complex configuration and coding required if one part of the BPMS stack is replaced with an existing component. In more open environments, lighter-weight components can augment or replace the BPMS functionality when it is necessary to improve performance or create functionality.

Enterprise application development environments typically include standard code repositories, version control and test tools, which are essential for tracking and testing changes to applications. Many BPMS lack robust versioning capabilities, and are completely without automated testing tools, since these are not seen as essential when non-technical analysts are the target audience.

Another driver for using developer-friendly BPM, where developers can use a familiar environment and language (such as Eclipse

and Java) to perform most of their work, is that little retraining of developers is required. Developers need to learn the modeling notation (e.g., BPMN) and the capabilities of the BPMS that can be invoked from their applications, but do not need to learn a completely new application development environment.

Finally, there is less vendor lock-in to the BPMS, since the vendor-specific knowledge required is relatively small compared to the knowledge of the existing corporate-standard development languages and frameworks. This reduces the risk of selecting a specific BPMS tool, allowing it to be more easily replaced in the future.

Making Developer-Friendly BPM Work For Everyone

Developer-friendly BPM is not about returning to an age of IT fiefdoms and waterfall development methods. Business analysts, who have a much better understanding of the business, should control the overall form of the process model that is visible to the business, while IT, which has a much better understanding of the integrating technologies, ensures that all the pieces fit together and the infrastructure to support it is sound. It's important to recognize that process modeling isn't just a matter of drawing boxes and lines: it's about understanding the mapping between the real world and a process model, especially when that real world process includes many events and systems. Consider the task of calling a service from a process: it's easy to hook up the inputs and outputs of a service to a task in a process, but the challenge is understanding the world of services that are available to be called, and the impact and costs of calling any particular one on the underlying systems. The business analyst knows that something automated needs to happen at a particular point, but a technical architect or developer should be deciding how that is done.

BPMN presents a good opportunity for non-technical analysts to be involved in modeling processes, although it is often unnecessarily restricted. One attempt at isolating business users and analysts from the complexities of process models was to create conformance classes within the BPMN standard: subsets of the element types visible to less-skilled modelers, in particular to limit the set of available events from about 50 (in the entire standard) down to a more manageable number. Although it's true that some of the less-commonly used event types do not need to be available for non-technical modelers, limitations should be made based on usage

patterns, not specific elements. For example, an interrupting timer boundary event on an activity is a concept that is completely understandable regardless of technical skill: if a task is not completed after a certain amount of time, then divert it to an exception-handling path. This event type, however, is not available in the subset of event types that is usually presented for non-technical modelers.

The key is not to restrict the element types available to analysts, only to have developers add them into the same process later; rather, analysts should be able to model using all elements that make sense to them for modeling the business. That business-created process model should be executable and drive analytics, so that what a business user sees in a monitoring dashboard is their own process model, not an unrecognizable augmented technical version of that model. The executable business version of the process model can't do everything that is required for a complex process application, but it can still execute as the main driver of the business process while linking to technical business processes and code via messages and services in a manner that is completely transparent to, and understandable by, business users. This gives control over executable business processes back to the business, while still allowing developers to create technical processes for orchestration, code for transactions, and lightweight links between them. The technical process model and code are not refinements or replacements for the business process model: they're more like services invoked by it.

Summary

When creating complex, core business process applications, "zero code" doesn't mean "zero developers", and a proprietary BPMS development environment can hinder enterprise developers.

Lightweight developer-friendly BPM that integrates with existing enterprise environments can be a better fit with corporate development standards, requires less training for developers and has less vendor lock-in.

Making developer-friendly BPM work without returning to the era of lengthy requirements documents and waterfall development requires using executable process models where it makes sense: for business-IT alignment and technical process orchestration, but not for general code development unrelated to process.

About the Author

Sandy Kemsley is an independent analyst and process architect specializing in business process management (BPM). She consults to end-user organizations and BPM vendors globally, writes a popular BPM blog at www.column2.com and is a featured conference speaker on BPM and related topics.

Sponsor

This paper was sponsored by camunda, whose product camunda BPM is a lightweight, open source platform for Business Process Management. It is dedicated to software developers and their typical infrastructure, while providing business-IT alignment during process design and runtime using the BPMN 2.0 - standard.

See: http://camunda.com/bpm