Editor: **Cesare Pautasso**
University of Lugano
c.pautasso@ieee.org

Editor: **Olaf Zimmermann**
University of Applied Sciences
of Eastern Switzerland, Rapperswil
olaf.zimmermann@ost.ch

Creating a Low-Code Business Process Execution Platform With Python, BPMN, and DMN

Dan Funk

From the Editors

The visual Business Process Modeling and Notation (BPMN) offers nonprogrammers a "low-code/no-code" language for describing and automating their processes. The author of this edition of the "Insights" column shares his experience on an open source project developing a BPMN execution engine in Python. Along the journey, the team valued user interest in features over compliance and standards coverage; learned that less is more when it comes to teaching and learning notations; and learned that sticking to software engineering principles and practices such as Don't Repeat Yourself, version control, and test automation pays off in their development context—just like in any other one.—*Cesare Pautasso and Olaf Zimmermann*

WE JUST COMPLETED the beta release of a development effort on a Python/BPMN workflow system. We have learned some hard lessons, and as we look forward to a successful launch this summer, it is a good time to reflect.

The Problem

Our software, developed for the University of Virginia's School of Medicine (UVA SOM), streamlines the submission process to UVA's Institutional Review Board for Health Sciences Research. UVA SOM requested a general-purpose workflow system, a tool that could handle frequent tweaks and adjustments, because this domain is dominated by change. As many researchers are aware, a university's review board is the perfect storm—a collision of science, bureaucratic processes, legal liability, and the progression of professional careers.

Our Approach

We researched many options for a method to meet UVA SOM's requirements and chose Business Process Model and Notation (BPMN) because it is an open standard with a rich and comprehensive specification and wide adoption. BPMN includes a powerful and robust workflow diagram standard that looks a little like flow charts. BPMN 2.0 is particularly notable for its ability to be directly executed, meaning that it can not only describe requirements but actually implement them. We determined that a system based on BPMN could offer a unique solution to UVA's needs. The use of

Digital Object Identifier 10.1109/MS.2022.3212033
Date of current version: xxxxx

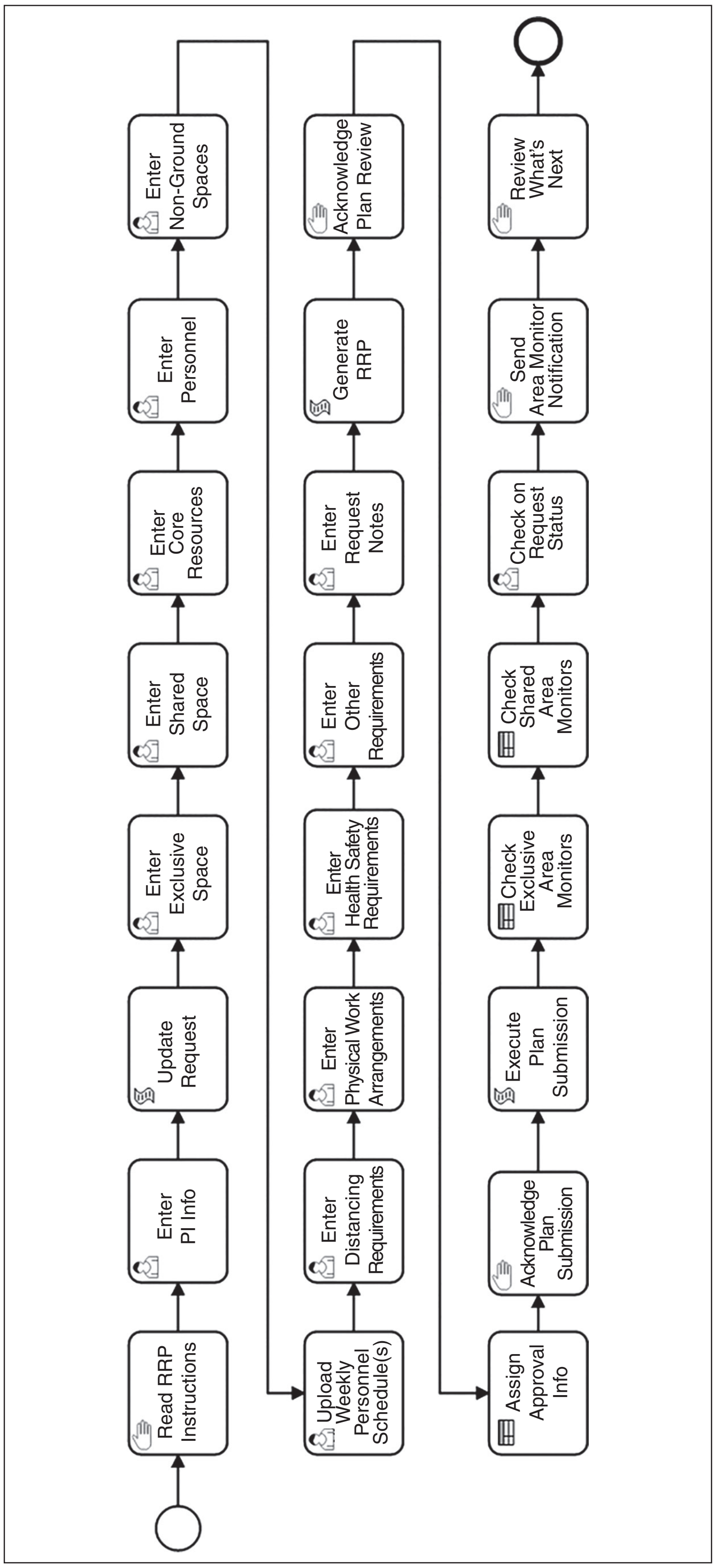


FIGURE 1. This model describes the Research Recovery Program (RRP), a submission and approval process for bringing a research lab back online during the first summer of the pandemic. PI: principal investigator.

easy-to-understand diagrams creates critical transparency for decisions that directly affect many different departments, researchers, and reviewers A well-composed BPMN diagram reminds stakeholders of competing requirements and alternate perspectives. Rules, compromises, and long-established traditions remain open to investigation. Displaying the process openly and clearly invites iterative development as everyone seeks to find new efficiencies while coping with inevitable change.

For reference, Figure 1 shows the first BPMN model we used in a production system. We'll talk more about this diagram and what we have done since later in the article. Each box represents an activity or task. Those with a person icon are completed by a human being. The wavy paper denotes Python scripts, and the spreadsheet denotes a Decision Model and Notation (DMN) decision table, which we will also discuss later.

BPMN offered us the opportunity to create a "low-code" environment, a place where citizen developers (used here to mean domain experts who may have little prior software development experience) can grow from drawing simple diagrams to developing their own Python code to meet business requirements, all within the cradle of a maintainable and transparent software architecture. While "low-code" is a popular buzzword at the moment, we believe BPMN offers something better. We prefer the term *As Much Code As Needed* since there is not a technology cliff awaiting adopters but rather a seamless path to more complex development.

We found that the BPMN community was well grounded in Java with many implementations.[1]

Python, however, offers some significant advantages over Java when working with BPMN. Both Python and BPMN have low-sloped, but long, learning curves. Writing a first program in Python is quick and immediately rewarding. Drawing a BPMN diagram feels immediately intuitive and simple as well. For both Python and BPMN, there is no obvious brick wall, no point at which the technology would prohibit further progress. BPMN is highly expressive and deep. Python is dynamic and broadly applicable. There is good reason to believe that coupling these technologies provides a strong chance of success if given the right audience.

What We Learned

Over the last two years, we have learned a lot about BPMN. We will divide our lessons learned into three areas. First, we will cover BPMN and DMN from our client's perspective and how people receive a BPMN-enabled application. Second, we will cover some thoughts about citizen developers. Finally, we will dig into the technical challenges of building BPMN software in Python.

For context, these are lessons we learned as we created more than 60 individual business processes that managed areas such as document collection and organization; compliance reviews; financial calculations; submissions; and approvals. There were 207 individual BPMN diagrams (many shared across workflows as call activities) and 97 DMN tables. Completing all the individual process instances can take many months, with some individual process instances running for weeks. There may be dozens of concurrent processes, but this particular project is neither processor nor memory intensive.

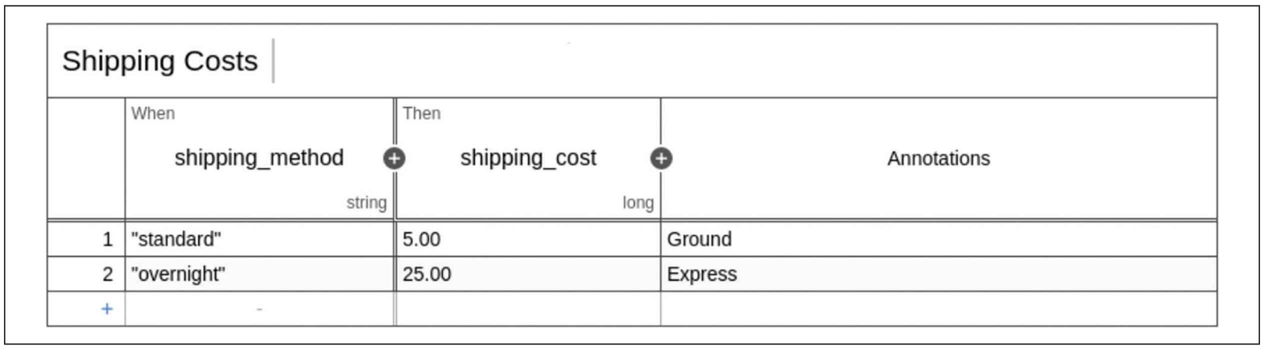
Introducing BPMN and DMN to the Client

We ran into issues early on as we attempted to train individuals to use BPMN to address business challenges. We were building a new interpreter and asking our client to run their BPMN diagrams on that interpreter as it was being developed. We were blessed with some very patient people with excellent communication skills. Now that the core BPMN and form-processing components are fully functional, BPMN training should be much easier. However, all the pain of working in the tool as it was being built helped create a truly useful tool. The alternative would have been to build blindly to the specification without the push and pull of real-world needs.

We have found the adoption of DMN decision tables (spreadsheet-like tables that map inputs to outputs) to be much higher than the BPMN diagrams. These decision tables offer a near-seamless transition for business analysts as they move from creating spreadsheets that merely describe requirements to ones that implement requirements. A DMN decision table allows us to define a decision by mapping a set of inputs (test conditions) to a set of outputs (when each condition is met).

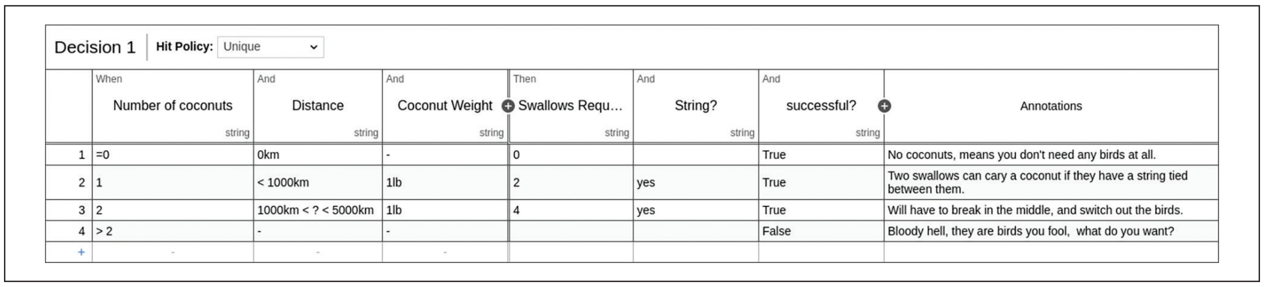
DMNs can be very expressive, but to offer a simple example, Figure 2 shows a DMN that maps shipping types (the "When" column) to shipping costs (the "Then" column). Figure 3 shows a slightly more complex example of a DMN table.

As we move into the future, we will introduce DMN tables earlier in the training program but always within the larger context of BPMN to avoid a technology wall (see the "Supporting the Citizen Developer" section). Based on our experience so far, I suspect that we will find that a portion of the audience is most comfortable working within the confines of these DMN tables. A well-authored


Shipping Costs

	When shipping_method string	Then shipping_cost long	Annotations
1	"standard"	5.00	Ground
2	"overnight"	25.00	Express
+			

FIGURE 2. When conditions match the content in the "When" column (left column), the content of the "Then" column (middle column) is produced, such that "standard" shipping outputs a cost of US$5.00.



Decision 1 | **Hit Policy:** Unique

	When Number of coconuts string	And Distance string	And Coconut Weight string	Then Swallows Requ… string	And String? string	And successful? string	Annotations
1	=0	0km	-	0		True	No coconuts, means you don't need any birds at all.
2	1	< 1000km	1lb	2	yes	True	Two swallows can cary a coconut if they have a string tied between them.
3	2	1000km < ? < 5000km	1lb	4	yes	True	Will have to break in the middle, and switch out the birds.
4	> 2	-	-			False	Bloody hell, they are birds you fool, what do you want?
+	-	-	-				

FIGURE 3. In the classic movie *Monty Python and the Holy Grail*, there is a running argument about how coconuts could show up in England, and one proposal is that they are carried by swallows. This table provides rules about how many swallows it would require to transport a given number of coconuts a certain distance, thereby showing a DMN example while removing all pleasure from the original joke.

BPMN diagram can encapsulate frequently shifting business requirements into this decision table format, which requires far less cognitive load; it is possible to quickly edit the DMN and then get back to other work without reinvesting mental effort and time in mastering the BPMN standard.

Supporting the Citizen Developer

We are highly conscious of walking people into a technology wall. I am reminded of climbing up the side of a mountain at Machu Picchu with my then eight-year-old son, where it became evident that an uncomfortable step to one person can be a nearly insurmountable cliff to another. Moving from DMN decision tables and then to BPMN diagrams and to small Python scripts are not seamless steps, but it is possible to take them without massive shifts of perspective because these are all first-order entities in the BPMN framework. Our recommendation would be to train with these steps in mind.

Briefly, here is how I would introduce concepts to a new client—through a pizza ordering system demonstration.

Step 1: Introduce the Very Basic BPMN Tools. They don't have to understand everything initially—just start with the simplest case and allow BPMN to describe and define the scope of the workflow process. In this case, we need to get a pizza order and figure out how much it costs (Figure 4).

Step 2: Show How to Gather Information From Users. "What do you want on your pizza?" is pretty clearly a user task, which will mean gathering information from the end user—likely through some sort of web form. Show people how to define a form using a form builder. In this case, it would be a list of toppings that we could collect in a toppings variable, which will be passed on to the next task.

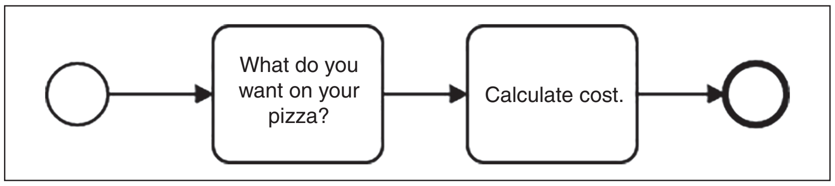


FIGURE 4. A very simple BPMN diagram, but it is all that is required to get started.

Step 3: Introduce DMN. As quickly as possible, demonstrate how to encapsulate business decisions into decision tables. Calculating cost is a great opportunity to do so.

Step 4: Run It. Provide immediate feedback that the tool works and indeed solves the real (albeit limited) problem.

Step 5: Repeat, Adding Complexity. BPMN is a powerful and expressive notation, with the ability to describe parallel execution, branching logic and collaborations. We recommend that you introduce these concepts in the context of resolving other business problems. For example, daily specials and half-off on Tuesdays might be good opportunities to add branching logic (gateways). Sending the final order to the kitchen's ordering system (a service task) or modeling the work in the kitchen (lanes) are other potential learning activities. The trick is to start small; encapsulate the business rules in decision tables; and iterate to solve increasingly complex problems. The most important concept here is not to expect mastery of BPMN at any point.

BPMN and Python

As described previously, Python was important to our project.

SpiffWorkflow is one of the few libraries written in Python that supports the parsing and execution of BPMN diagrams and was easily the most robust implementation we could find. SpiffWorkflow's source code is hosted on GitHub (https://github.com/sartography/SpiffWorkflow). When we began, active development had slowed on the project, but the community was large, with more than 200 forks and 1,000 stargazers (followers), and bug fixes were still coming in. We were able to build a prototype of a workflow system using SpiffWorkflow that was able to execute basic BPMN diagrams, including user tasks, script tasks, gateways, and subprocesses. The BPMN that we created online using the BPMN.io editor could be parsed and understood by SpiffWorkflow. We had a good prototype, but we soon realized that SpiffWorkflow was far from ready. We have since put a lot of effort into improving the documentation of SpiffWorkflow—with detailed example code to help others where we struggled. Please see ReadTheDocs (https://spiffworkflow.readthedocs.io/) for more information.

Just a few months into the project, we repurposed our code to create an emergency application for UVA to allow research labs to reopen over the late summer of 2020 in the heat of the pandemic. It was far from pretty, but we were able to stand the full approval process up in a few weeks (and a few sleepless nights). The BPMN diagram for this project is shown as an example in Figure 1. That initial diagram was completely linear. This was due to some limitations we encountered with parallel gateways and the persistence of state. We found many issues with the library (then at version 0.6) that made that first production deployment difficult. We have since resolved these issues and many others.

We now make extensive use of many more BPMN constructs as we added support for them in SpiffWorkflow. Figure 5 is a more recent workflow for running a contract through the finance committee for oversight.

During the project, we spent hundreds of hours improving the library. We have made more than 40 pull requests into the open source repository, which amounts to half the total pull requests since the project's inception in 2010. Not only were these accepted, but the previous maintainer added our team, and eventually, turned over maintenance of the project to us. We added support for DMN tables (multi-instance tasks, lanes, and events) while improving the execution environment, error handling, and serialization (that is, the persistence of state). We added these features as we discovered a clear and immediate need in our work for UVA. We never augmented the library as a blind effort toward meeting the specification, but we always ensured that we did work within the specification as we added new features. In the process, we have read the 500-page BPMN specification dozens of times and have benefited greatly from two excellent textbooks by Bruce Silver on BPMN[2] and DMN[3] that cover best practices in this domain.

We didn't have to do this work. We could have picked up the open source JBPM or a commercial offering from Trisotech, Flowable, or Camunda, which are mature and support a large portion of the BPMN specification. I feel certain that we would have taken a different approach if we had not found the SpiffWorkflow library, with its active community and amenable original maintainer.[4] Thankfully, we are finding that other people see the value in SpiffWorkflow as well, and we are collaborating with several other companies to continue to expand and develop this library.

There were some wonderful surprises in store for us as we worked to improve SpiffWorkflow, which included the following:

- The presence of a powerful and well-maintained open source BPMN editor makes it possible to author these diagrams and even embed the editor into our web applications (thanks BPMN.io).
- The BPMN standard is great for branching logic, such as complex questionnaires (that is, "you can skip this next set of questions as your research is just on UVA grounds, but we do need you to submit this on-grounds request to the building committee") that might branch off in different directions under specific circumstances. The implementation with BPMN and SpiffWorkflow was immediate, clean, and easy. It made our first efforts delightful.
- BPMN pools and lanes are an elegant way to model and implement approval processes. The diagrams are intuitive, and the implementation is clean.
- DMN tables are straightforward and relatively easy to implement. They are the perfect interface for business analysts.

There were also some very deep pitfalls, especially with data management. For small diagrams, it isn't an issue, but as the diagrams become larger, spanning many files, subprocesses, call activities, and decision tables, it becomes a significant

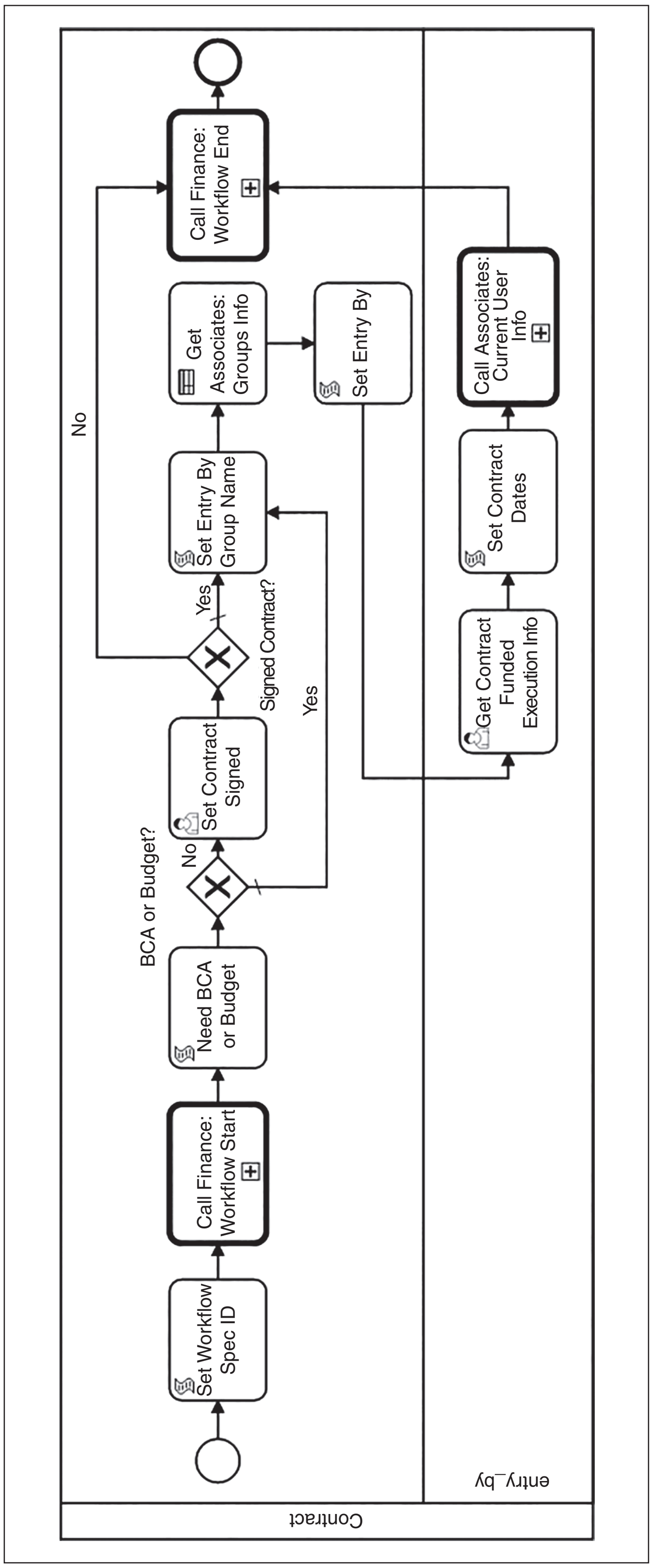


FIGURE 5. The diagram demonstrates the use of lanes to pass work off between different stakeholders and call activities (the bolded squares) to call out to other reusable BPMN diagrams. The exclusive gateways (marked with an "X") allow researchers to skip the irrelevant parts of the process when possible. There are user tasks (marked by people) and script tasks (marked with a paper icon), and a DMN table reference (the spreadsheet icon) is also represented here. This is still only a subset of the language elements available. BCA: billing coverage analysis.

problem. The BPMN specification speaks only briefly about DataObjects and DataStores and offers little consideration of variable scope. How data are scoped can have an enormous impact on the behavior of a diagram. (Data scoped to the process are simple but not at all thread safe; data scoped to a task are clean, but then, how do tasks communicate?) We researched implementations from existing open and commercial products, but the handling of data varied wildly; was deeply nuanced and complex; and was always mired in implementation details.

Currently, we allow data to follow the sequence flows, using a pipe architecture, similar to the Unix command line. What comes out of one task naturally flows into the next, and each task has full control to modify the data as it sees fit. This works except for *call activities*, which are calls to externally defined reusable BPMN diagrams. Such compositions require something more complex, such as *method signatures*, a way to define the required inputs and outputs. (For additional information, please see our detailed documentation on data modeling.[5]) Such signatures can be handled using BPMN's data input and data output, and we are building this into the next release of SpiffWorkflow.

Extensibility

Both communicating with external services and data analytics were handled by injecting custom scripts into our Python execution engine. To make a known application programming interface (API) call, the BPMN developer would simply call a predefined method from within a script task. These predefined methods must be maintained as a part of the application, which is not ideal. In an upcoming release of SpiffWorkflow, we will provide tools for building API calls using a service task that requires no custom software development outside of the BPMN model. This effort is underway and should be released by the time this article is published. Interactions with external services can at times become very complex (API keys, asynchronous calls, and authentication handoffs), so we are also actively developing a means to manage these security concerns as well.

To track important events, we made a logging function available to script tasks as well. In this way, the developers of the BPMN models could call out specific events during the execution of a workflow. Other functions also available to script tasks are able to query on these logs to determine things like the time between two events (for instance, "it took three days from the time the approval was requested to it being accepted"). The ability to query the logs meant that the BPMN developers could build their own custom reports. The BPMN flow would display a form that accepted certain parameters; it would then query the logs to find the information, and finally, display the data using Python's Jinga templates to render the report as an HTML table. We are now creating general-purpose reporting systems as well to answer questions that were not premeditated.

Some Successes

Although it was painful at times to build a tool while simultaneously trying to use it for a specific project, it allowed us to learn some valuable lessons and to make a few really good choices. Most of our good choices were simply applying good software development practices to BPMN, such as the DRY principle; using a version control system; and automating tests.

We worked hard to ensure that any code embedded in the BPMN diagrams could be evaluated as Python, and it was definitely the right thing to do. BPMN uses conditional logic for gateways; its script tasks must be evaluated; DMN tables contain many expressions; and forms often need expressions for things like whether to show or hide an input field. Our battle to ensure that all of these things were consistently evaluated in the same way was hard but enormously beneficial. When building "low-code" applications, one must ensure that their users are learning only one programming language, not many.

We created a testing framework for quick feedback on BPMN diagrams, and while it was not as comprehensive as a suite of unit tests, we were able to build a basic tool to sanity-check our BPMN diagrams for errors by executing them within a test harness—completing forms with random data (or no data at all if the field wasn't required) and executing some, if not all, paths through the workflow to ensure that the process would work. We plan on carrying this further, allowing us to define assertions and connect them to the BPMN model in some way. We can also set breakpoints on a specific task in a BPMN diagram to allow easier debugging at that point of execution. What we learned early on was that even an imperfect testing framework–if it offers instant feedback–pays enormous dividends in productivity and reduced frustration. "Low-code" modelers require a lively[6] programming environment to make progress, just like the rest of the development world.

A recent refactor moved our BPMN diagrams out of the database and onto the file system, where they are maintained as a Git repository. This has a host of benefits. It raises the BPMN diagrams to their rightful status as collaboratively built software. The transfer of diagrams between instances (that is, from development to staging to production) can be handled with fine-grained control over precisely which changes are merged and deployed. With BPMN diagrams in a Git repository, it is now trivial to pull down the latest production version, find the issue, and push up a fix (While we have not put it to use yet, this would be an ideal place to utilize the BPMN Diffing library[7] with BPMN.js to visualize the differences[8] rather than looking at the XML.) It's so useful that it almost feels like I'm pointing out the obvious. So I'll at least say it succinctly: if it smells like code, keep it in a version control system.

Future Aspirations

Over the last two years, we learned much about BPMN. Perhaps most insightful is that we have not learned to hate it. Conversely, we found ourselves embracing its many facets and seeing tremendous potential in this standard, and we plan to work with it for many years to come.

We never added a component to SpiffWorkflow that was not precipitated by a real business need. So it is interesting to see what components we did add and which have yet to assert themselves.

Please check out our existing unit tests (you can find some of our test BPMN files in this subdirectory on GitHub[9]), 521 of them in total, which cover many core features and edge cases. In the future, we hope to build a set of tests around the BPMN Model Interchange Test Suite,[10] which will help clarify our overall feature set. In the meantime, we took a close look at Section 2.0 (Conformance) of the BPMN 2.0 specification.[11] We looked at 70 core aspects of the specification and categorized them into seven major areas (Table 1). There are nuances that are not addressed with these numbers. This is not meant to be a thorough study of our implementation but a brief self-assessment in the hopes that it sheds light on where the specification helped us address our needs versus areas where we could not make an effective implementation work.

Support for data structures stands out at a lackluster 12.5%. The reasons are twofold. First, the standard is somewhat vague on how data should be scoped and passed between activities and tasks. Second, the implementation of these in the BPMN.io editor we depend on is incomplete since Camunda went its own route for data management (likely because of the first point).

We have not implemented the service task (under "Tasks and activities" in Table 1) as of yet, but the need is there. We were able to accomplish calls to external systems using script tasks, but the asynchronous behavior and the added security and clarity of service tasks are highly attractive and are currently under active development in our GitHub repository.

We have some limitations in our support of events and messages. We've found that we often compose processes using call activities and shared repositories of reusable workflow processes that we call *Libraries*. I suspect that as we build larger and more complex systems, we will become more reliant on messages and less so on call activities. We have recently commenced heavy development work on messages, also available on GitHub.

The more important truth in these numbers is how well the standard did address our needs. When we needed to allow people to complete steps out of order, the parallel gateway was there. When we needed to build approval processes, the lanes were there. When we needed to interrupt that approval process midstream because of a change of heart, messages and events were there. When we implemented these BPMN solutions, we were often astonished at their power and versatility.

Having now done it, would we recommend that others create their own BPMN interpreter/compiler? They would need a very compelling reason. It is inglorious work. We hope that having this robust open source Python-based BPMN interpreter will open the door for exploration into the future evolution of BPMN. We hope that we have saved others from the grueling effort of the baseline implementation so that they might enjoy the more delightful opportunity to experiment in new frontiers.

For those building a Python application that would benefit from an embedded BPMN system, I would certainly encourage them to consider using the SpiffWorkflow library. We plan to continue our contributions, and we are about to undertake the development of a whole new set of libraries that will make integrating SpiffWorkflow into custom applications even easier.

Table 1. A rough overview of the features defined in the BPMN 2.0 specification versus features implemented in the SpiffWorkflow Library.

Data	Total	Implemented	Percentage
Tasks and activities	11	9	81.82%
Flow control	9	9	100%
Pools and lanes	3	3	100%
Events and messages	35	26	74.29%
Visual appearance	1	1	100%
Data	8	1	12.5%
Miscellaneous	2	1	50%
Grand total	**69**	**50**	**72.46%**

This is an exciting time for SpiffWorkflow. We are collaborating with four companies on enhancements to the library and have received a similar number of recent pull requests from external contributors. These are small numbers, but they reflect major investments in time and energy from previously unaffiliated sources. SpiffWorkflow has also received 300 additional stars (likes) on GitHub in the last year as well as tickets and feature requests from around the world, creating an exponential growth curve over the course of the last 10 years.[12] There is momentum here that we are building upon, and we believe that the greatest years of innovation lay ahead—in what our users will create with this library.

References

1. "List of BPMN 2.0 engines." Wikipedia. Accessed: Jun. 5, 2022. [Online]. Available: https://en.wikipedia.org/wiki/List_of_BPMN_2.0_engines
2. B. Silver, BPMN Method and Style. Aptos, CA, USA: Cody-Cassidy Press, 2012.
3. B. Silver, DMN *Method* and *Style*: The Practitioner's Guide to Decision Modeling with Business Rules. Aptos, CA, USA: Cody-Cassidy Press, 2018.
4. "Samuel Abels." GitHub. Accessed: Jun. 25, 2022. [Online]. Available: https://github.com/knipknap
5. "Putting it all together." SpiffWorkflow. Accessed: Jun. 25, 2022. [Online]. Available: https://spiffworkflow.readthedocs. io/en/latest/bpmn/synthesis.html#examining-the-workflow-state
6. S. L. Tanimoto, "A perspective on the evolution of live programming," in *Proc. 1st Int. Workshop Live Program.*, May 2013, pp. 31–34, doi: 10.5555/2662726.2662735.
7. "bpmn-io/bpmn-js-differ." GitHub. Accessed: Aug. 5, 2022. [Online]. Available: https://github.com/bpmn-io/bpmn-js-differ
8. "bpmn.io." GitHub. Accessed: Aug. 5, 2022. [Online]. Available: https://demo.bpmn.io/diff
9. "sartography / SpiffWorkflow." GitHub. Accessed: Oct. 10, 2022. [Online]. Available: https://github.com/sartography/SpiffWorkflow/tree/main/tests/SpiffWorkflow/bpmn/data
10. "bpmn-miwg/bpmn-miwg-test-suite." GitHub. Accessed: Oct. 10, 2022. [Online]. Available: https://github.com/bpmn-miwg/bpmn-miwg-test-suite
11. "Business process model and notation," Object Management Group, Needham, MA, USA, 2010. Accessed: Oct. 10, 2022. [Online]. Available: https://www.omg.org/spec/BPMN/2.0/
12. "Star History." Accessed: Oct. 10, 2022. [Online]. Available: https://star-history.com/#sartography/SpiffWorkflow

ABOUT THE AUTHOR

DAN FUNK is the owner of Sartography, Staunton, VA 24401 USA, a software consulting firm that focuses on supporting academic research. Contact him at dan@sartography.com.