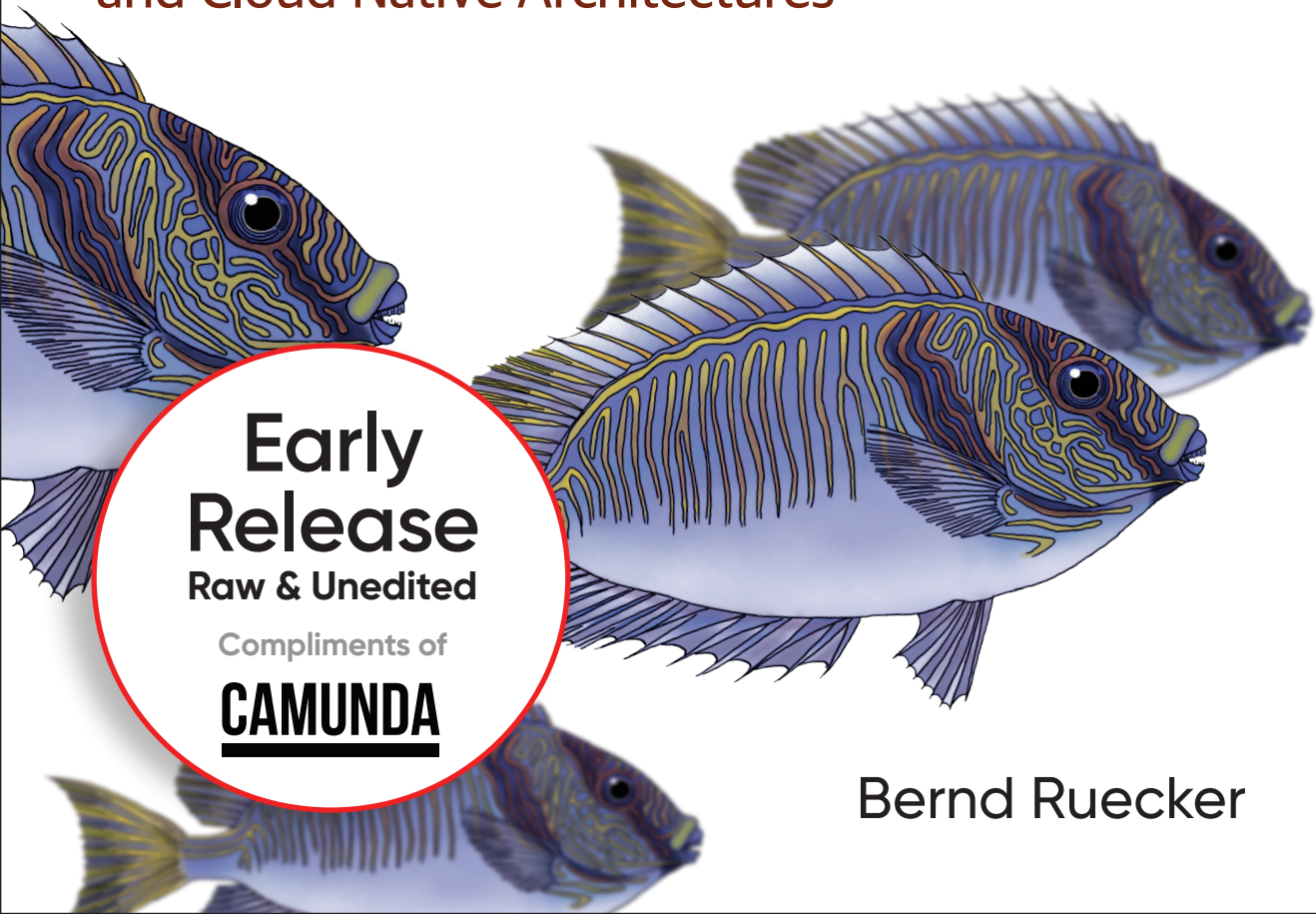
O'REILLY®

Practical Process Automation

Orchestration and Integration in Microservices and Cloud Native Architectures



Early Release
Raw & Unedited

Compliments of
CAMUNDA

Bernd Ruecker

Practical Process Automation

Orchestration and Integration in Microservices and Cloud Native Architectures

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

Bernd Ruecker

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Practical Process Automation
by Bernd Ruecker

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Melissa Duffield
Development Editor: Michele Cronin
Production Editor: Deborah Baker
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

April 2021: First Edition

Revision History for the Early Release

2020-09-01: First Release
2020-10-21: Second Release
2021-01-08: Third Release
2021-02-17: Fourth Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781492061458* for release details.

978-1-492-06138-0

[LSI]

CHAPTER 1
Introduction

Let's get started! In this chapter you will:

- Gain a basic understanding of what I mean by process automation.
- Understand specific technical challenges when automating processes.
- Learn what a workflow engine can do and why this provides a ton of value.
- Read about how business and IT can collaborate when automating processes.
- See how modern tools differ very much from BPM and SOA tooling from the past.

Process Automation

In essence, a process (or workflow) simply refers to a series of tasks that need to be performed to achieve a desired result.

Processes are everywhere. As a developer, I think of my personal development process as being able to manage certain tasks that go from an issue to a code change that is then rolled out to production. As an employee, I think of optimizing my process around handling emails, which involves techniques that prioritize them quickly and keep my inbox empty. As a business owner, I think of end-to-end business processes, e.g., to fulfill customer orders known as "order to cash". And as a back-end developer, I might also think of remote calls in my code, as this involves a series of tasks, especially if you consider retry or cleanup tasks because a distributed system can fail at any time.

Processes can be automated on different levels. The main distinction is if a human controls the process, if a computer controls the process or if the process is fully automated. Here's an example that highlights these different levels of automation.

During my civil service I helped organize the meals-on-wheels deliveries to elderly people in their homes. There was a daily process going on to handle the meal orders, aggregate a list of orders that went to the kitchen, package the meals, and finally ensure that all the orders are labeled correctly so they will be delivered to the correct recipients. In addition to that, there was the delivery service itself. When I started, the process was completely paper-driven, and it took an entire morning to accomplish. I changed that, leveraging Microsoft Excel in order to automate some tasks. This brought the processing time down to about 30 minutes—so it was a lot more efficient. But there were still physical activities involved, like packing and labeling the food as well as driving to the recipient's home.

More importantly, the process was still human-controlled, as it was my job to press the right buttons and show up in the kitchen at the appropriate times with the appropriate lists. Only some tasks were supported by software.

In my last hospital visit I chatted to the staff about how the meal preparation worked. The patients were required to fill out a paper card to mark allergies and meal preferences, and this information was typed into a computer. Then the IT system was in charge of timing and transporting that information to the right place, and it needed to be done in an automatic fashion. People still played a role in the process, but they did not steer it. This was a computer-controlled, but not fully automated process.

If you take this example even further, today there are cooking robots available. If you would add these robots to the above process, it would be possible to task the computer with not only automating the control flow, but also the cooking tasks within that. This moves the process closer to a fully automated process.

As you can see, there is an important distinction between the automation of the control flow between tasks, and the automation of the tasks itself.

1. *Automation of the control flow:* The interactions between tasks are automated, but the tasks itself might not. If humans do the work, the computer controls the process and involves them whenever necessary, for example using tasklist user interfaces. This is known as human task management. In the previous example, this was the humans cooking the food. This is in contrast to a complete manual process that works because people control the process passing paper or emails around.
2. *Automation of the tasks:* The task itself is automated. In the previous example, this would be the robots who cook the food.

If you combine both, the automation of the control flow and the tasks you end up with *fully automated* processes, known as straight-through processing (STP). These processes only require manual intervention if something happens beyond the expected normal operations.

While there is of course an overall tendency to automate processes as much as possible, there are specific reasons that motivate for automation:

High number of repetitions
: The effort put into automation is worthwhile only if savings exceed the cost of development. Processes with a high volume of executions are excellent candidates for automation.

Standardization
: If processes are barely structured and run differently all the time, it is hard to automate them.

Compliance conformance
: For some industries or specific processes there are strict rules around auditability, or even rules to follow a documented procedure in a repeatable and revisable manner. Automation can deliver this and provide relevant data right away, in high quality.

Need for quality
: Some processes should produce results in continuous quality. For example you might promise a certain delivery speed with customer orders. This is easier to achieve and retain with an automated process.

Rich in information
: Processes that carry a lot of digitized information are better suited to automation. If physical objects have to be touched, automation is more difficult.

Automating processes can be achieved by different means, as further examined in "Limitations of Other Implementation Options" on page 95. But there is also special software that is dedicated to process automation. As mentioned in the preface, this book will focus on the latter, and especially look at workflow engines.

Automating processes does not necessarily mean to do software development or use some kind of workflow engine. It can be as simple as leveraging office tools, Slack or Zapier to automate tasks triggered by certain events. For example, every time I enter a new conference talk in my personal spreadsheet, it triggers a couple of automated tasks to publish it on my homepage, the company event table, our developer relations Slack channel and so forth. This kind of automation is relatively easy to implement, even by non-IT folks in a self-service manner, but of course limited in power.

In the rest of this book I will *not* focus on these office-like workflow automation tools. Instead, I will look into process automation from a software development and architecture perspective.

In order to understand how to automate processes with a workflow engine, let's quickly jump into a story that helps you understand real-life developer problems it can solve.

Wild West Integrations

Imagine the back-end developer Ash who gets tasked with building a small back-end system for collecting payments via credit card. This doesn't sound too complex, right? Ash starts right away and designs a beautiful architecture. They talk to the folks doing order fulfillment and agree that providing a REST API for the order fulfillment service is the easiest option to move forward. So Ash goes ahead and codes that thing.

Half way through a colleague walks in, and looks at their whiteboard, where the beauty of the architecture is captured. The colleague casually asks: "Ah, you are now using that external credit card service X. I used to work with it, too. We had a lot of issues with leaky connections and outages back then, did that improve?"

This question hits Ash by surprise. This expensive SaaS service is flaky? That means their nice, straightforward code is too naive! But no worries, Ash adds some code to retry the call when the service is not available. After chatting a bit more, the colleague tells Ash that their service suffered from outages that sometimes lasted hours. Puh—so Ash needs to think of a way of retrying over a longer period of time. But darn it—this involves state handling and using a scheduler, so they decide to not tackle this right away but just add an issue to the backlog in the hopes that the order fulfillment team can sort it out. For now ash simply throws an exception when the credit card service is unavailable and crosses fingers that all will work out well.

Two weeks into production a different colleague from order fulfillment walks over, alongside the CEO. What the heck? It turns out Ash's system raised a lot of credit card service unavailable errors, and the CEO is not happy about the amount of orders not being fulfilled: this issue has resulted in lost revenue. Ash tries to act immediately and asks the order fulfillment team to attempt retrying the payments, but they have to iron out other urgent problems and are reluctant to take over responsibilities that should be within Ash's service (and they are totally right to do so as you can read about in Chapter 7).

Ash has to promise the CEO that they can fix this situation, and get something live ASAP. Back at their desk, Ash creates a database table called **payment** and gives it a column **status**. Now they insert every payment request there and give it a status **open**. On top of that Ash adds a simple scheduler that checks for open payments every couple of seconds and processes them. Now the service can do stateful retries over longer periods of time. This is great. Ash calls the order fulfillment folks and they discuss the changing API, as payments are now processed asynchronously. The original REST API will hand back HTTP 202 (Accepted) and Ash's service could either call back the

fulfillment service, send them some message or let them periodically poll for the payment status. The teams agree on the polling approach as a quick fix—as in the end Ash just needs to provide another REST endpoint to allow querying the payment status.

The change gets rolled out to production and Ash is happy to have the CEO off their back. But unfortunately the peace doesn't last too long. Next, Ash sees a caravan of people coming into their office, including the director of operations. They tell Ash that no order can be shipped because no payments are successfully being taken. What? Ash makes a mental note to add some monitoring to not be surprised by these situations in the future. But for now Ash turns towards their monitor and looks into the database. Oh no, there are a huge amount of open payments piling up. Digging a bit into the logs Ash discovers that the scheduler was interrupted by an exceptional case and crashed. Dang it.

Ash puts the one poisoned payment that interrupted the whole process aside, restarts the scheduler, and see that payments are being processed again. Relieved, Ash vows to look into the database twice a day to make sure this does not happen again. The next thing they do is hack a small script to periodically look at the table and send them an email whenever something unusual happens. Ash even discovers that they can add some mitigation strategies for the exceptional case to that script. Great!

After all these stressful weeks Ash plans to go on vacation. But it turns out that their boss isn't too happy about Ash leaving because nobody except Ash actually understands the tool stack that they just built. And even worse, the boss instead pulls out a list of additional requirements for the payment service, as some business folks heard about the flaky credit card service and want more in-depth reports about availability and response times. They also want to know if the agreed-on service level agreement (SLA) is actually being met and want to monitor that in real time. Gosh—now Ash has to add report generation on top of a database that they actually never really wanted in the first place. Figure 1-1 shows the resulting mess in its full beauty.

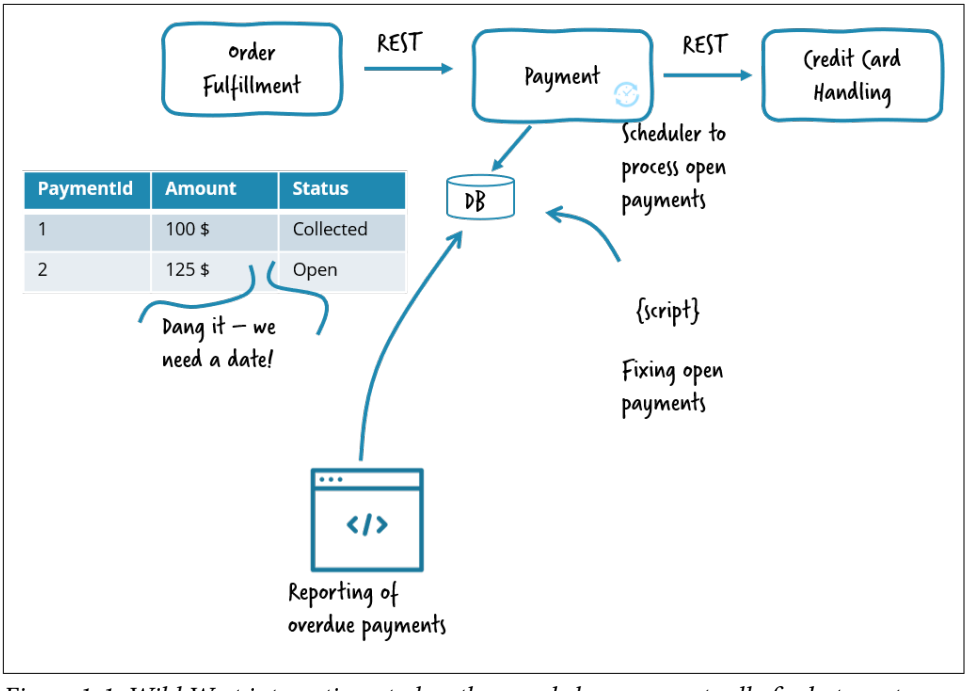


Figure 1-1. Wild West integration at play, the usual chaos you actually find at most enterprises

Unfortunately, Ash just used a far too common approach to automate processes that I call wild west integration. It's an ad-hoc approach to creating systems without any kind of governance. It is very likely that such a system doesn't serve the business as a whole well.

Here are more flavors to wild west integration:

Integration via database
: a service accesses some other services' database directly in order to communicate, often without the other service knowing it.

Naive point-to-point integrations
: Two components communicate directly with each other, often via REST, SOAP or messaging protocols, without properly clarifying all aspects around remote communication.

Database triggers
: Additional logic is invoked whenever you write something to the database.

Brittle tool chains
: For example moving comma separated text files (CSV) via FTP.

The story above leads to a home-grown workflow engine. Keeping the current state, scheduling retries, reporting on the current state or operating long-running processes are build-in capabilities of a workflow engine. So, you should leverage existing tools instead of writing your own engine. And there is really nothing to gain. Even if you think that your project doesn't need the additional complexity of a workflow engine, you should always give that a second thought.

Home-grown workflow engines have severe shortcomings. The most obvious problem is that it is hard to understand how the business process is really implemented. The other not so obvious problem is that the processes have requirements around persistent state handling. With your bespoke solution you have to code this state handling into the components itself, leading to a lot of subsequent requirements that need to be tackled, basically the core features of a workflow engine.

Coding processes without a workflow engine typically results in complex code; state handling ends up being coded into the components themselves. This makes it harder to understand the solutions since the process itself can't be easily understood.

Workflow Engines And Executable Process Models

So what is the alternative to home-grown workflow engines? You can use an existing workflow engine (such as one of the open source products contained in the curated list on the website of this book).

A workflow engine automates the control of a process. It allows you to define and deploy a blueprint of your process, the *process definition*, expressed in a certain modeling language. With that process definition deployed you can start *process instances* and the workflow engine keeps track of their state.

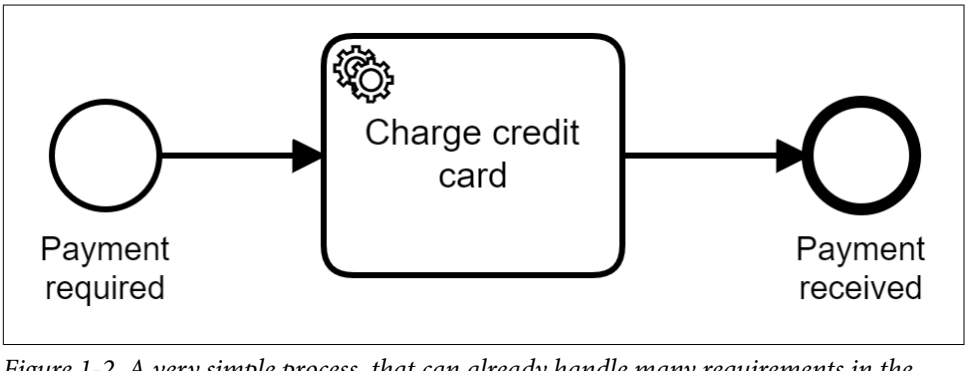


Figure 1-2. A very simple process, that can already handle many requirements in the credit card example

Figure 1-2 shows a process for the payment example introduced earlier. The process starts when a payment is required, as indicated by the first circle in the process model, which is a so-called start event marking the beginning of a process. It then goes through the one and only task, which is a so-called service task, indicated by the cog wheels. This service task will implement the REST call to the external credit card service. You will learn how this can be done exactly in Chapter 2. For now, simply imagine that you write some normal programming code to do this, which I call glue code. After that task, the process ends in the end event, the circle with the thick border.

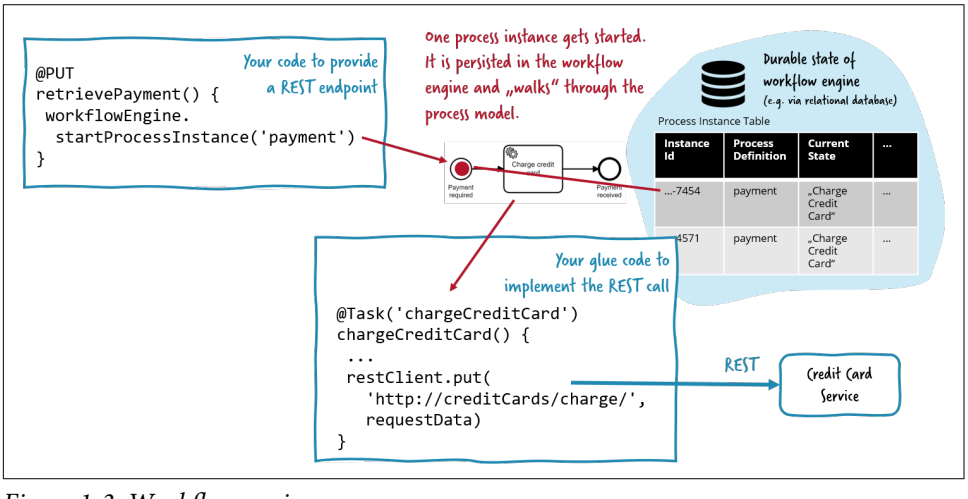


Figure 1-3. Workflow engine

Figure 1-3 visualizes with some pseudocode how you can use this process model to implement payments. First, you will write some code that reacts to something in the outside world, for example a call to the REST endpoint to collect payments. This code will then use the workflow engine API to start a new process instance.

This process instance is persisted by the workflow engine; Figure 1-3 visualizes this via a relational dabatase. You can read about different engine architectures, persistence options and deployment scenarios later in this book.

Furthermore, you will write some glue code to charge the credit card. This code acts like a callback and will be executed when the process instances advances to the task to charge the credit card, which will happen automatically after the process instance was started. Ideally, the credit card payment is processed right away and the process instance ends afterwards. Your REST endpoint might even be able to return a synchronous response to its client. But in case of an outage of the credit card service, the workflow engine can safely wait in the task to charge the credit card and trigger retries.

We just touched on the two most important capabilities of a workflow engine:

- Keep the state persistent, which allows waiting,
- Schedule things, like the retries.

Depending on the tooling, the glue code might need to be written in a specific programming language. Some products also allow arbitrary programming languages. So if you decide to cleanup your wild west code, you could probably even reuse big parts of your code and just leverage the workflow engine for state handling and scheduling.

Of course, many processes go far beyond that simple example. When retrieving payments, the process model might solve more business problems. For example, the process could react on expired credit cards and wait for the customer to update their payment information as visualized in Figure 1-4.

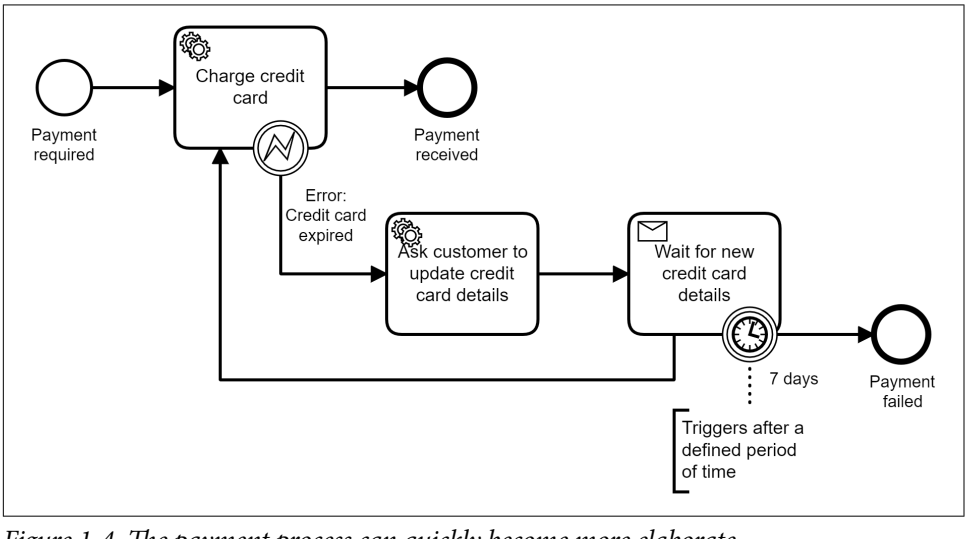


Figure 1-4. The payment process can quickly become more elaborate

But so far, the payment process is more an integration process, which is not the most typical use for process automation. I like starting with it, as it helps technical audiences to understand core workflow engine capabilities. But let's examine a more typical business process in the next section.

A Business Scenario

Let's look at a typical (but imaginary) project. ShipByButton Inc. (SBB) is a tech startup. They provide a small hardware button. Whenever it is pressed, one specific item is ordered. For example, you could put this button next to your washing powder, and when you see that the powder is almost empty, you just press that button, and

one box of washing powder will then be ordered and shipped to you (if this reminds you of the Amazon Dash button, this might be simply coincidence ;-)).

SBB wants to automate its core business process, which is order fulfillment. An elaborate discussion of the different roles and their collaboration is sketched in "A Typical Project" on page 197. For now, let's just say SBB starts with drawing out the process relating to the physical steps involved. Then they work their way down to the level of detail that can be automated using a workflow engine, helped by the fact that the process modeling language BPMN is universal regardless of the level you apply it at.

The resulting process model is shown in Figure 1-5.

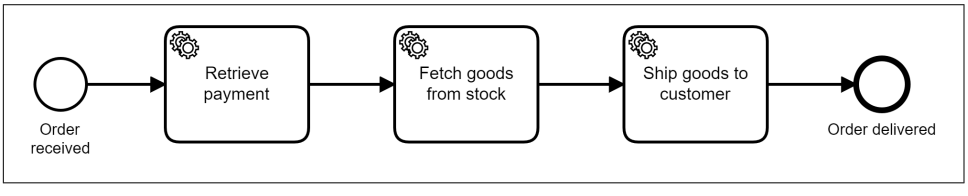


Figure 1-5. End-to-end business process that is subject to automation

This is of course a bit simplified, as in real life you have more exceptional cases; e.g., if payment cannot be retrieved or goods are out-of-stock.

You can see, that the first task of this process might invoke the payment service just described with the wild west integration. This is a typical scenario when applying microservices, as you will learn later in this book.

Modeling business processes often leads to an interesting by-product: unexpected insights. In a customer's scenario close to SBB we found that the "business people" did not actually know exactly what the "on the warehouse floor people" were doing. The visual process model not only helped to identify but also to resolve this problem.

Long-Running Processes

Let's conclude that process automation is very diverse. While it is often about enterprise, end-to-end business processes like order fulfillment, account opening, or claim settlement, it can also help in much more technical use cases around orchestration and integration, as noted in the credit card example.

All these examples share one commonality though: they are about *long-running processes*. Long-running refers to processes that take minutes, hours, weeks or months to complete. Handling long-running processes are where workflow engines excel.

The term "long-running" is a bit confusing sometimes, as it basically means to wait for something to happen, for example for other components to respond, or simply for humans to do some work. This is why workflow engines needs to handle durable state, as introduced earlier.

Another way to look at it is that long-running behavior is required whenever logic *crosses boundaries*.

When I say boundaries, this can mean very different things. If you call a remote service, you cross the boundary of your local program, your local OS, and your local machine. This leaves you responsible for dealing with problems around the availability of the service or added latency. If you invoke another component or resource, you also cross the technical transaction boundary. If you integrate components of other teams you cross organizational boundaries, which means you need to collaborate with these people more. If you involve external services like from a credit card agency, you cross the boundary of your own company. And if you involve people, this crosses the boundary beetween automatable and not-automatable tasks.

Managing these boundaries not only need long-running capabilities, but also requires you to carefully think about the *sequence of tasks*. Especially failure scenarios and the proper business strategy to handle them need serious discussion. And you might face regulatory requirements around data security, compliance or auditing. These requirements motivate for graphical process visualizations which will be covered in depth in Chapter 11. This allows technical folks to consult with the right non-technical people to solve these challenges.

Modern systems have more and more of these boundaries, as there is a big tendency to move away from monolithic systems towards fine-grained components, like services, microservices or functions. And systems are assembled out of a wild mix of internal applications and services simply consumed in the cloud.

Business Processes, Integration Processes and Workflows

To summarize, you can automate business processes as well as integration processes. The boundary between these categories is often not sharp at all, as most integration use cases have a business motivation. This is why you don't find "integration processes" as a separate category in this book. Instead, "Model or Code?" on page 59 will show you that many technical details will end up in normal programming code, and not in a process model. And "Extract (Integration) Logic Into Subprocesses" on page 213 will explain, that you can also extract some portions of the process model into child models. This allows you to push too technical details on its own granularity level, which helps to keep the business process understandable.

Furthermore, you might have noticed that I used both the terms process and workflow. Truth be told, there is no common, agreed-on understanding of the difference

between process automation and workflow automation. Many people use these terms interchangeably, while others argue that business processes are more strategic and workflows are more tactical artifacts, whereas workflows can be modeled and probably even executed on a workflow engine (which is sometimes also called process engine). Process models can also be called workflow models, and some standards interchange between of either term. Neither is right or wrong. I often recommend adjusting the terminology to that works well in your environment.

However, for this book I had to make a choice, and I simply went with what I feel most comfortable with. As a rule of thumb,

- Business process automation is *what* you want to achieve. It is the goal. It is what business people do care about. I will use process (or business process) in most cases.
- Workflow is used whenever I talk about the tooling, which is about *how* it is implemented. So for example, I will talk about a workflow engine, even if this will automate process models.

In real-life, I sometimes adjust these rules. For instance, when talking to technical folks about the implementation I might prefer the term workflow, workflow engine, or sometimes even orchestration engine or Saga, depending on the context (you will understand the latter terms when you progressed further in this book).

Business-IT Collaboration

The collaboration of business stakeholders and IT professionals is crucial for the success of modern enterprises. Business stakeholders understand the organisation, the market, the product, the strategy and the business case for each project. They can channel all of that into requirements, features and priorities. IT, on the other hand, understands the existing IT landscape and organization, constraints and opportunities as well as effort and availabilities. Only by collaborating, both "sides" can win.

Unfortunately, different roles often speak different languages. Not literally, both might communicate in English, but in the way they phrase and understand things.

Putting the business process in the center of this communication helps. It makes it much easier to understand requirements in the context of a bigger picture. It avoids misunderstandings that happen when you discuss features in isolation.

Visual process models facilitate this conversation, especially if they can be understood by business and IT. All efficient requirement workshops I saw were filled with people from business *and* IT.

A common example is that business folks underestimate the complexity of requirements, but at the same time miss easy picks. A typical dialogue goes like this:

- "Why is this small button so much effort?"
- "Because we need to untie a gigantic knot in the legacy software to make it possible! Why can't we just make a change over here and reach the same result?"
- "What, wait, we can change that over there? We thought that it is impossible."

With the right mindset and a good collaboration culture, you will not only progress faster, but also end up with better solutions and happier people. Process automation and especially visual process models help you. Chapter 10 will explain this in much more detail.

Business Drivers and the Value of Process Automation

Organizations apply process automation to

- Build better customer experiences,
- Get to market faster (with changed or completely new processes, products or business models),
- Increase business agility,
- Drive operational cost savings.

This can be achieved by the promises that come with the prospect of process automation: increasing visibility, efficiency, cost-effectiveness, quality, confidence, business agility, and scale. Let's look at some of these briefly.

Business processes provide direct visibility to business stakeholders. For example, a business person cares about the sequence of tasks, such as ensuring that payment is collected before shipping, or knowing what is the strategy for handling failed payments. This information is needed to truly understand how the business currently runs and performs. The data that process automation platforms provide lead to actionable insights, which is the basis for process optimizations.

Enterprises care about the efficiency and cost-effectiveness of their automated processes as well as quality and confidence. An online retailer might want to reduce the cycle time of their order fulfillment process, meaning that a customer will receive a parcel as fast as possible after hitting the order button. And of course, retailers also don't want any orders to fall through the cracks in the system, leaving them not only with a missed sale, but also an unhappy customer.

Some business models even rely on the possibility of fully automating processes; it is crucial for companies to make money, or deliver responses as fast as expected, or scale their business.

Business agility is another important driver. The pace of IT is too fast to really anticipate any trend properly, so it is important for companies to build systems that can react to changes. As the CIO of an insurance company recently said to me: "We don't know what we will need tomorrow. But we do know that we will need something. So we have to be able to move quickly!" Concentrating on building systems and architectures in such a way that makes it easy to adopt changes is crucial to the survival of many businesses. Process automation is one important piece, as it makes it easier to understand how processes are currently implemented, to dive into discussions around changes and to finally implement them.

Not Your Parents' Process Automation Tools

If process automation and workflow engines are a great solution for certain problems, why does not everybody apply them? Of course, some people simply don't know about it. But more often, people either had bad experiences with bad tools in the past, or they only have a vague association with terms like workflow or process automation and think they relate to old-school document flows or proprietary tool suites, and think these aren't helpful. Spoiler alert: this is wrong!

In order to overcome these conceptions it is good to be aware of the failures made in history, as it allows you to free your mind to adopt a modern way of thinking about process automation.

A Brief History of Process Automation

Let's look back in history. The roots of dedicated process automation technology began around 1990 when paper-based processes began to be guided by document management systems. In these systems, a physical or digital document was the token and workflows were defined around that document. So, for example, the bank account opening application form was scanned and moved automatically to the people that needed to work on it.

You can still spot these document-based systems in real life. I recently saw a tool being used with a lot of phantom PDF documents being created just to be able to kick off workflow instances that are not based on a real physical document.

This category developed further into human workflow management tools that were centered around human task management. They had their zenith around 2000. This time, you did not need documents to start a workflow. Still, these systems were built to coordinate humans, and not to integrate software.

Then service oriented architecture (SOA) emerged around the year 2000, as a an alternative to large monolithic ecosystems where traditional enterprise application integration (EAI) tools did point-to-point integrations. The idea was to break up functionality into services that are offered in a more or less standardized way into the

enterprise, so that others can easily consume it. One fundamental idea of SOA was to reuse these services and thus reduce development efforts. Tools emerged that were either rooted in EAI but added human task capabilities, or human workflow products that added integration capabilities.

Around the same time, business process management (BPM) gained traction as a discipline, taking not only these technical and tooling aspects into account, but also the lessons around setting up scalable organizations and business process re-engineering.

These developments are summarized in Figure 1-6.

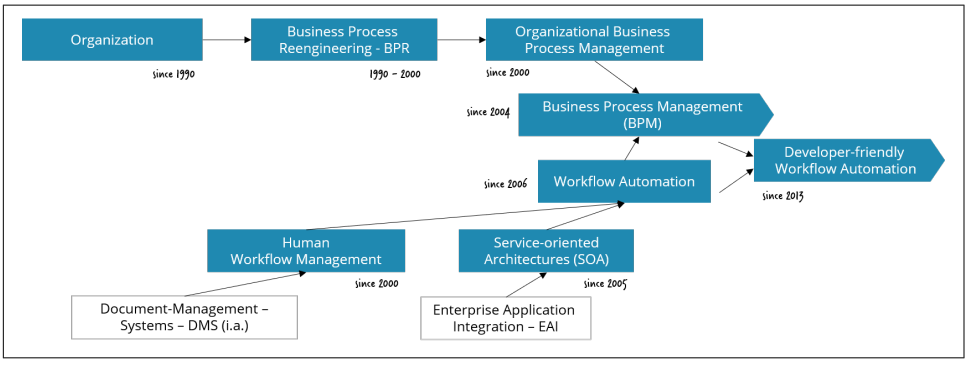


Figure 1-6. Historical development of disciplines

Process automation was a hyped topic in the BPM and SOA era. Unfortunately, there were some major flaws that disappointed many projects and individuals for the following reasons: BPM was too detached from developers and the tools were too vendor-driven, too centralized and too focused on low-code. Let me explain.

BPM in the Ivory Tower

BPM as a discipline includes methods to discover, model, analyze, measure, improve, optimize, and automate business processes. In that sense, it is a very broad topic. Unfortunately, many BPM initiatives were too detached from IT. For a long time, the folks doing BPM worked in silos, not considering how processes were really automated within the given IT infrastructure. This led to process models that could not function in real life, and yet these models were given to the IT departments to "simply" implement. Unsurprisingly, this did not work very well.

Centralized SOA and the ESB

In an instance of unfortunate timing, SOA collided with the high times of very complex technologies like simple object access protocol (SOAP), which made it difficult for any development team to offer or consume any other service. This opened up the space for tool vendors. Since SOA initiatives were typically very centrally organized

and governed, it brought the big vendors into the game, and they sold very expensive middleware that was placed in the heart of many companies in a top-down approach. The tooling was called an enterprise service bus (ESB) which was a messaging system in its core with multiple tools around it to connect services or transform data.

Looking back at SOA from today's perspective, it is easy to highlight some of the shortcomings:

Centralized
: SOA and ESB tools were typically installed as centralized systems and were operated by their own teams. This very much led to situations where you not only had to implement and deploy your own service, but also interact with the SOA team to deploy additional configuration into these tools, which caused a lot of friction.

Alien to the development process
: Tools broke the development workflow, making automated testing or continuous integration/continuous delivery (CI/CD) pipelines impossible. Many of the tools did not even allow for automated testing or deployment.

Vendor-driven
: The vendors overtook the industry and sold products before best practices existed, which forced practices into many companies that simply did not work.

Mixed infrastructure and business logic
: Important business logic often ended up in routing procedures that were deployed on the middleware, leaving it without clear ownership or responsbility. Now different teams implemented various aspects of logic that better belong in one place.

But how does this relate to process automation? Great question! SOA typically came in tandem with BPM suites.

Misguided BPM Suites

BPM suites (BPMS) were standalone tools that included a workflow engine in its core and tools around it.

Like ESBs, these suites were vendor driven. They were deployed as centralized tools that were introduced from the top down. In these environments a central team took care of the platform, and this team often was the only group capable of deployment. This dependence on single teams led to a lot of problems.

It's worth mentioning that that BPM suites emerged during a time when most companies were still running software on physical hardware—automated deployment pipelines weren't really a thing then.

The Limitations of Low Code

BPM suites came with the promise of "zero code," which was later branded as "low code." The idea is as simple as it is appealing to business stakeholders: develop processes without IT being involved so a non-technical person can create an executable process model without writing programming code.

Low-code approaches involve heavyweight toolings that allow these non-developers to build processes by dragging and dropping pre-built elements. Sophisticated wizards allow to configure them, so that you should be able to build solutions without writing any source code.

This approach is still sold as desirable by advisory firms and BPM vendors. And the low-code approach indeed has its upsides. There is a shortage of developers at the moment, so many companies simply don't have the resources to do proper software projects as they would like to. Less tech-savvy people (referred to as "citizen developers" by Gartner) begin working on software projects and need these low-code approaches.

A low-code approach might work for relatively simple processes. But it definitely falls short on complex business processes or integration scenarios. What I regularly found is that low-code products do not deliver on their promise and less tech-savvy citizen developers cannot implement core processes themselves. As a result, companies have to revert back to their IT departments and ask them to assign professional software developers to finish the job. Those software developers need to learn a proprietary, vendor-specific way of application development. Developing this skill takes a long time, and it's often a frustrating experience. As a result, there is a lack of sufficiently skilled software developers within in the organization, which forces companies to look for outside resources.

Those outside resources are system integrators that partner with the BPM vendor and provide consultants certified by that vendor. Those consultants are either not as skilled as promised, too expensive, or simply not available, and often all of that at the same time.

Furthermore:

- You can't use industry best practices to develop software solutions, like automated testing or frameworks that you might need for integration or user interfaces. You can only do what the vendor has foreseen, as it is hard or even impossible to break out the preconceived path.
- You are often blocked from open-source or community-driven knowledge and tool enhancements. For example, instead of being able to pick up a code example from GitHub, you instead have to watch a video tutorial on how to use the proprietary wizard to guide you through the low-code interface.

- Tools are typically very heavyweight and do not easily run in modern virtualized or cloud-native architectures.

These unfortunate dynamics caused a lot of companies to no longer engage process automation tools, even though not all approaches involve this type of proprietary software or low-code development.

Instead of replacing software development with low-code process automation, the focus should be on bringing software development and process automation together!

It is important to understand that agility does not come from implementing processes without the help of developers, but because different stakeholders can understand and discuss graphical models.

As soon as you can combine process automation with "normal" software development practices, you gain development efficiency and quality, you allow normal developers to work on these jobs and you have a whole universe of existing solutions available helping you out on any kind of problems. Workflow vendors might still prebuild support for certain integrations which then helps to reduce the effort to build solutions.

Moving Past Old-School BPM Suites

The good news is that there are now a lot of really useful and lightweight workflow engines available, that integrate well with typical development practices and solve common problems.

This new generation of tools is most often open-source or provided as a cloud service. They target developers and support them in the challenges described earlier in this chapter. They deliver real value and help our industry to move forward.

The Story of Camunda

I always like to back this whole development with the story of the company I cofounded: Camunda, a vendor that—as marketing nowadays says—reinvented process automation. As mentioned in the preface, this book shall not be a marketing vehicle for the company, yet the story can help you understand the market development.

I started the company together with my cofounder in 2008. Back then, we started Camunda as a company providing consultancy services around process automation. We did a lot of workshops and trainings and thus had thousands of customer contacts.

This collided with the peak times of the old BPM and SOA ideas and tools. We could observe various tools in different companies. The common theme was that it did not work out. And it was not too hard to figure out the reasons. I described them earlier in this chapter: centralized, complex, low-code, vendor-driven.

So we experimented with the open-source frameworks of that time. They were much closer to developers, but they also could not cut it, mainly because they were too basic, lacked important features and required too much effort to build your own tooling around it.

At the same time, we collaborated on the development of the BPMN standard. BPMN (business process model and notation) defines a visual but also directly executable process modeling language.

And we saw a huge opportunity: a workflow engine that is developer-friendly, provided open-source and fosters business-IT collaboration by using BPMN.

We validated that idea with first customers and soon made a decision to pivot with the company. So in 2013 we transformed Camunda from a consulting firm into an open-source process automation vendor. Our tool was the complete opposite to the common low-code BPMS back then.

Today, Camunda grows fast and has hundreds of paying customers and countless community users. Many big organisations trust in the vision and even replace big vendor toolings throughout their companies. We accelerate growth globally, as process automation tooling is strongly needed. This is fueled by digitalization and automation programs as well as the trend to move towards more fine-grained components and microservices, that then need to be coordinated. In short: we are doing very well.

Technically, the Camunda workflow engine is engineered as you engineered applications at 2013. It is basically a library, built in Java, that used a relational database to store state. The engine can be embedded into your own Java application or run standalone and provide a REST API. And of course, there are a couple of additional tools to model or operate processes.

This architecture served Camunda very well and can handle most performance and scalability requirements of today. Nonetheless, a couple of years back, we developed a workflow engine in a completly different architecture, that nowadays is best described as being cloud-native. This workflow engine backs the managed service offering within Camunda Cloud and scales infinitely.

This enables the usage of a workflow engine in even more scenarios, which is a vision we've had in mind for a long time.

Conclusion

This chapter showed that process automation is a centerpiece of digitalization efforts. This makes workflow engines a vital building block in modern architectures.

Workflow engines solve problems around state handling and allow to model and execute graphical process models. This helps you to avoid wild west integration and fosters business-IT collaboration when automating processes. You saw a first example of a process model, directly executed on a workflow engine, something that will be explained further in the next chapter.

We have great technology available today, that is very different to old-school BPM suites. It is not only developer-friendly, but also very performant and scalable.

PART I
Fundamentals

This part will foster a general understanding of process automation with workflow engines.

Therefore Chapter 2 introduces workflow engines and the execution of process models on such an engine with a hands-on example.

Chapter 3 answers very practical questions on how to implement executable processses and connect them with other parts of your application. This gives you a solid understanding how process automation can work in real-life.

Chapter 4 dive into the various use cases process automation can be applied to, which is orchestration of humans, bots, software or decisions. This should give you a good idea how process automation is applicable in your context and which projects qualify for leveraging it. Note that Chapter 9 will look at further use cases of a workflow engine, namely how the workflow engine can be applied to solve certain challenges in distributed systems.

To conclude the fundamentals, Chapter 5 will give you reasons why workflow engines and BPMN are a great choice to automate processes. There you can read about alternative implementation approaches and process modeling languages.

CHAPTER 2

Workflow Engines and Process Solutions

After the introduction to process automation, this chapter will

- Introduce workflow engines and process solutions.
- Present a hands-on, executable example to make things concrete.
- Look at the developer experience when using process automation platforms.

The Workflow Engine

As you could see in the introduction, a workflow engine is the key component to automate the control flow of a long-running process.

If you wonder why you should use a workflow engine instead of hard coding processes or applying batches or data streams, you might want to peek into "Limitations of Other Implementation Options" on page 95.

Core Capabilities

The core technical capabilities of a workflow engine are:

Durable state (persistence)
: The engine safely remembers all running process instances, their current state and also audit data of historical actions. While this sounds easy, durable state is still a challenge to handle, especially at scale. Also it immediately triggers subsequent requirements around understanding the current state, which means you will need operations tooling. A workflow engine also needs to manage transactions, e.g. handling concurrent access to the same process instance.

Scheduling
: A workflow engine needs to keep track of timing and possibly escalate if a process gets stuck for too long. Therefore, there must be a scheduling mechanism that allows the engine to become active whenever something needs to be done. This also allows to retry tasks in terms of temporary errors.

Versioning
: Having long-running processes means that there is no point in time when there is no process instance running. Remember, that in this context "running" might actually mean waiting. Whenever you want to change the procedure and, for example, add another task to the process, you need to think about currently ongoing process instances. Most workflow engines support multiple versions of a process definition in parallel. Good tools allow migrating instances to a new version of the process definition, even in an automatable and testable manner.

These core features are visualized in Figure 2-1.

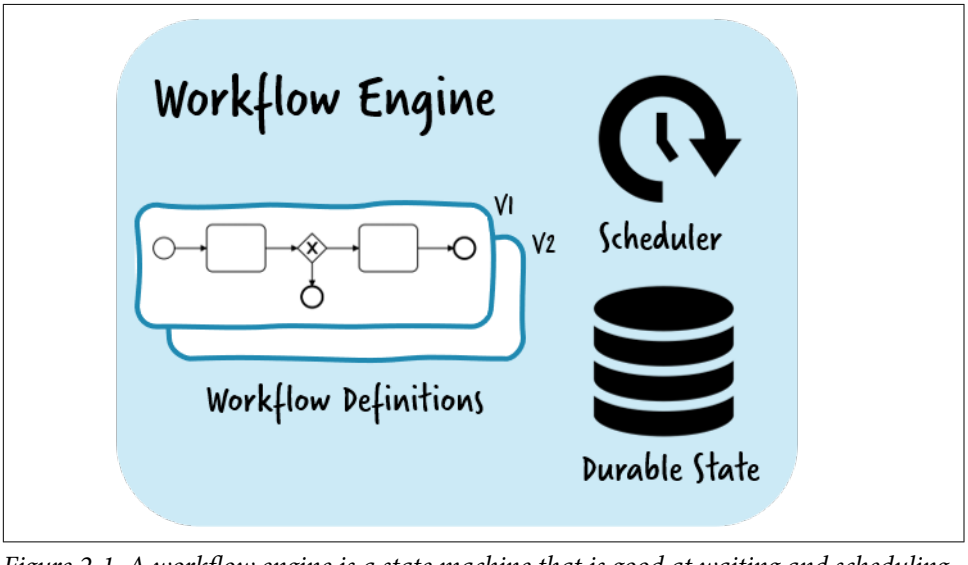


Figure 2-1. A workflow engine is a state machine that is good at waiting and scheduling

Using a workflow engine removes the burden from you to store state yourself, leverage bespoke scheduling mechanisms, and ultimately build your own workflow engine, as described in "Wild West Integrations" on page 4.

Of course, there are tradeoffs. The main disadvantage of using a workflow engine is that you introduce another component in your stack. This never comes for free. For example, you need to evaluate a specific tool, learn how to use it and sketch an architecture that makes use of it.

Typically, this pays off quite early, but of course this depends very much on your scenario. At this point in the book it is too early to discuss when it make sense to use a workflow engine. You first have to understand how such a workflow engine really works and how it influences your architecture, and then we will come back to that question in "When to Use a Workflow Engine" on page 115. To give you a sneak peak: the return on investment also depends on the investment, so if you choose very lightweight tooling that has a shallow learning curve, it can also help you to solve "smaller" problems.

Workflow Engines Are More Flexible than You Might Think

There are very different workflow engines with very different architectures and resource requirements available. Modern workflow engines are very lightweight and integrate well into your existing architecture, developer experience, and CI/CD pipelines. There are managed offerings in the cloud. And some of these workflow engines can horizontally scale and thus be used in high-load scenarios, like trading use cases where latency matters, telecommunication use cases with huge throughput or in retail use cases with high peak loads to be mastered.

Additional Features of Workflow Platforms

Alongside with these core capabilities, most workflow engines provide additional features. Good tools make these features optional or pluggable, which gives you the ability to choose if you simply want a super-lean workflow engine helping the developer, or also leverage some more tooling around that. This can also be a journey where you adopt more features when you see the need for them.

Typical additional features are:

Visibility
: Process models can be expressed graphically, either in relatively simple visualizations or powerful graphical languages. I will discuss this in detail in "Process Modeling Languages" on page 102. Having visibility of how the process is implemented is a huge communication gain and help for different roles, starting with the developer themselves ("How again did I implement this last year?") via operations ("What tasks happened before that incident?") to business stakeholders ("How is the process currently implemented? Can we improve this here?").

Audit data
: Workflow engines write a lot of audit data about what is going on, such as the timestamp when a process instance gets started, when it ends, when a certain task is entered, how often it needs to be retried, or incidents that happened. This data is highly valuable during operations, e.g., to recognize and understand a current

failure situation, as well as to understand the overall performance in order to improve the process itself. This data can also be used to provide important business dashboards where once there might have been a lack of transparency of work being done or the cost of processing.

Tooling
: Most tool stacks deliver not only the core engine, but also tools for graphical modeling, technical operations or business monitoring. "Typical Workflow Tools in a Project's Life Cycle" on page 38 will go into details.

Architecture

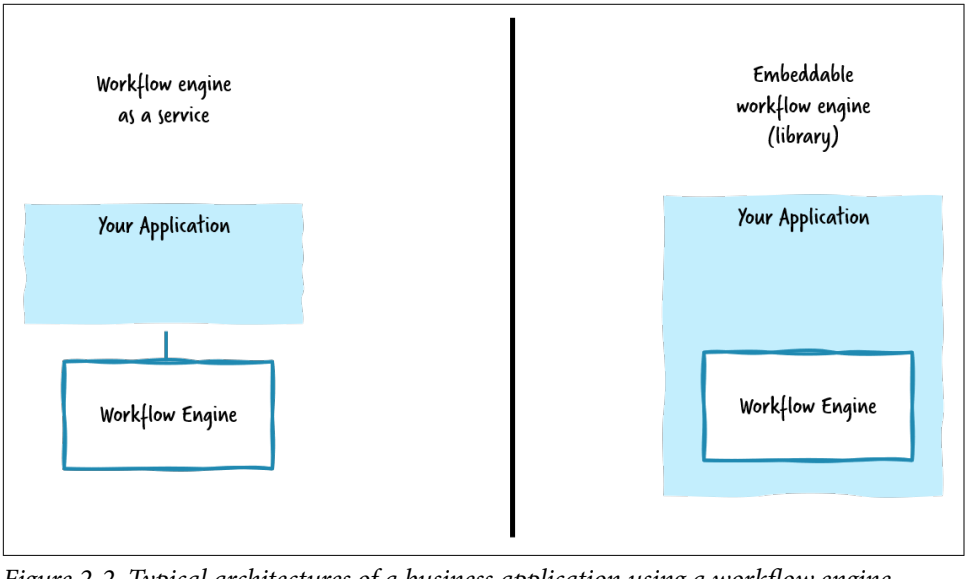


Figure 2-2. Typical architectures of a business application using a workflow engine

There are two basic options to run the workflow engine itself, visualized in Figure 2-2:

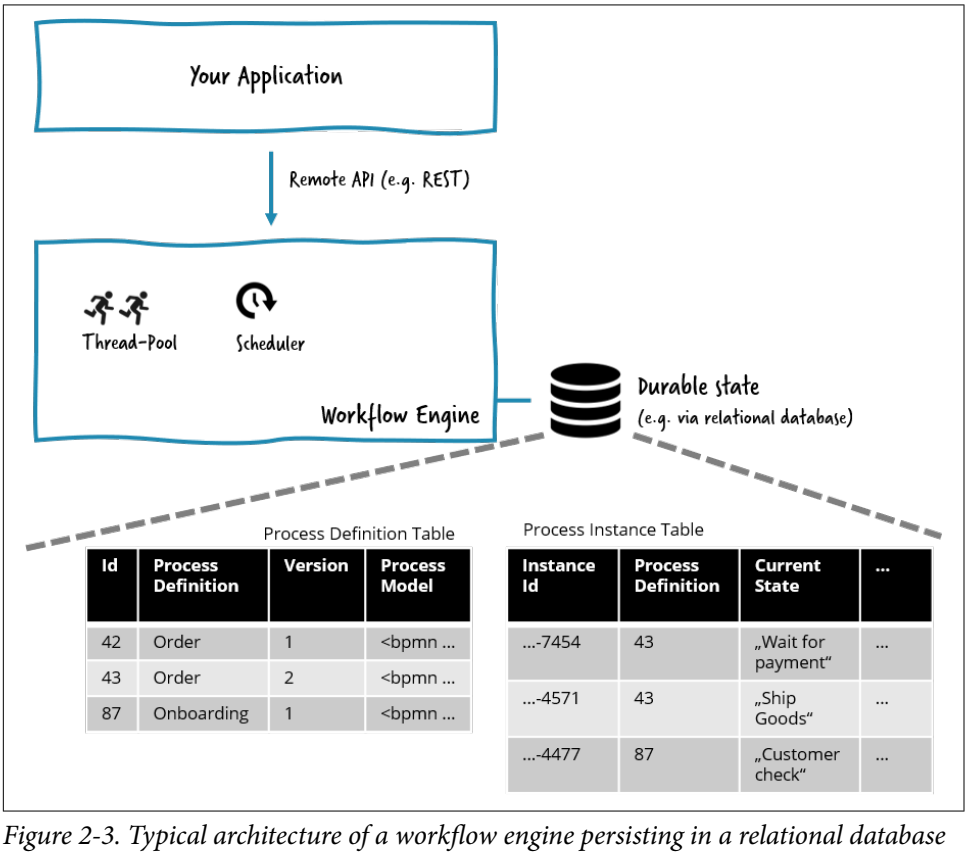
- The workflow engine is deployed as a service, meaning it is a self-contained application that is separate from your business application. This means that your business application talks remotely to the workflow engine (left-hand part of Figure 2-2).
- Some engines allow to be operated as library and thus run as part of your own application (right-hand part of Figure 2-2).

Having the workflow engine as a service should be considered the default nowadays. It allows you to isolate your glue code from the workflow engine, which can eliminate a lot of problems. I have learned this with support tickets as an example. With

embedded engines it takes a lot of effort to figure out how a customer embedded the workflow engine, and how that leads to the problems described.

As a bonus, running the workflow engine as a service allows you to use it from different programming languages. Modern environments make it easy to spin up such a workflow engine, e.g., via Docker, or consuming it as cloud service.

Internally, the workflow engine itself implements scheduling, thread handling and persistence. This is where there are big differences in the products. For the sake of understanding, let's assume that the workflow engine uses a relational database to store state. As Figure 2-3 visualizes, the workflow engine then keeps a record of all process definitions as well as all process instances. Whenever a process instance advances, the state is updated.



Id	Process Definition	Version	Process Model
42	Order	1	<bpmn ...
43	Order	2	<bpmn ...
87	Onboarding	1	<bpmn ...

Instance Id	Process Definition	Current State	...
...-7454	43	„Wait for payment"	...
...-4571	43	„Ship Goods"	...
...-4477	87	„Customer check"	...

Figure 2-3. Typical architecture of a workflow engine persisting in a relational database

Some workflow engines store state using something other than a relational database, e.g. they use a more event-sourced approach. The latter allows to scale beyond the limitations of a relational database, which allows modern engines to scale horizontally or to support high-throughput, low-latency or real-time applications. As a user

of the workflow engine, the way of storing the state is not your problem, but of course you need to understand the implications that affect you. If a relational database is used, it is important to know which products are supported as you need to operate that database. In case other ways of state handling are used, they might pose their own requirements that need to be checked.

One source of confusion is about threading. Whenever I say "waiting" or "long-running" in the context of a workflow engine, I don't mean that the workflow engine thread is blocked, waiting for something to happen. What the workflow engine does instead is storing the current state in the persistent database. Then, the workflow engine is finished. It returns the thread and does nothing any more.

But as the process instance state is kept in the database as long it is running, the process instance logically waits for something to happen; some event that will cause the workflow engine to load the state from the database again and resume processing. This could be a user pressing a button, which yields in an API call on the workflow engine, that completes the corresponding task. It could also be the engine scheduler that wakes up a process instance because some timer event is due.

A Process Solution

The process model is only one piece of the puzzle to automate a process. You will need to implement additional logic, typical examples being:

- Connectivity, e.g., to call REST endpoints or send AMQP messages
- Data handling and transformation
- Decisions, which path to take in a process model

The core workflow engine is not responsible for handling these aspects, even if most vendors will provide some out-of-the-box help for them. There is a thin line between convenience features you want to use and low-code features that you better stay away from, as described in "The Limitations of Low Code" on page 17.

For this book I assume that most of the additional aspects are handled where they can be handled best by developers: in programming code.

So for example, instead of using proprietary connectors of your workflow engine to implement an HTTP call, it might be easier to code this in Java, C#, NodeJS, or whatever language you are fluent in. This results in glue code, that is logically part of your process solution. Combining process models and code will be described in detail in "Combining Process Models and Programming Code" on page 54.

A workflow engine is also not responsible for storing business entities. This data should be stored by your application, and the workflow engine typically just refers to

it. So while it can technically store data alongside every process instance, the use of this capability should be limited to keeping references (ids).

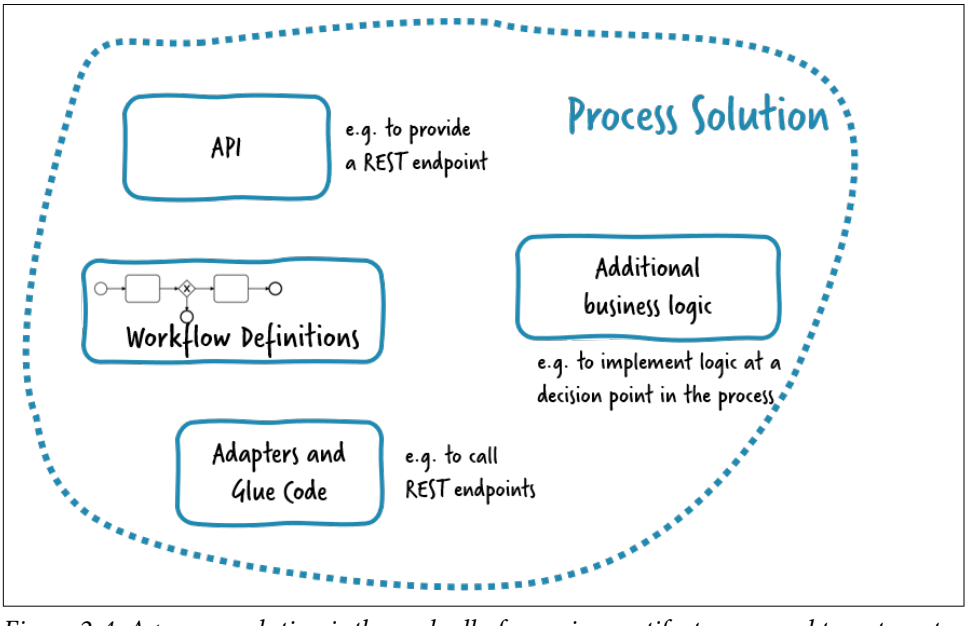


Figure 2-4. A process solution is the umbrella for various artifacts you need to automate a process, including but not limited to the process model

Figure 2-4 visualizes a process solution. Technically this could mean many different things, e.g., one project using Java and Maven, .NET Core, or NodeJS. But it could also mean a bunch of serverless functions logically bundled as a process solution.

Executable Example

Let's do a concrete example to make things more tangible. The source code for it can be found online and you are invited to play around with it yourself: *https://github.com/berndruecker/customer-onboarding-camundacloud-springboot* (TODO: Replace with *https://processautomationbook.com/* deep link).

For this example I use the following product stack:

- Java and Spring Boot
- Maven, so my Maven project equates the process solution
- Camunda Cloud, a managed workflow engine in the cloud

You can still relate many of the concepts and steps to other products. However, I need to choose a concrete stack to be able to show real source code.

The example will be extended later in the book and is about the onboarding of new customers in a small telecommunication company. The process model is shown in Figure 2-5.

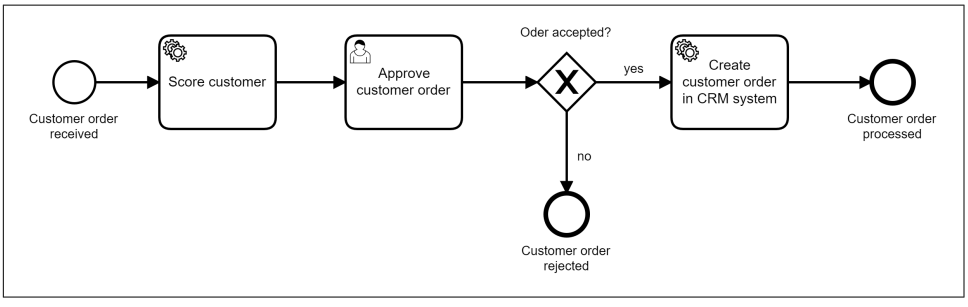


Figure 2-5. Process to onboard new mobile phone customers in a telecommunication company

When customers order a new mobile contract, a new instance of the process is started. The process first calculates a score of the customer using some Java code. It is doing this by a service task in the process, indicated by the cog wheels. The score is an input for the decision if the customer order will be accepted or not. This decision is made by an employee of the telecommunication company, it is explicitly not automated, as indicated by the human icon.

The result of that decision influences the path of the process instance at the upcoming XOR gateway, the diamond with the X. This gateway is a decision point, so the onboarding process instance either continues to automatically process the new customer order, or it ends. Of course, in a real scenario you would add some more tasks, e.g., to inform rejected customers.

Let's briefly explore what you need in order to bring this model to life. This is not about generating code from some business-owned model, but very much about taking exactly this process model and executing it on the workflow engine.

Customer onboarding shall be its own microservice with a REST API. This development project, visualized in Figure 2-6, will contain:

- The onboarding process model. Using BPMN, as it will be introduced in "Business Process Model and Notation (BPMN)" on page 45, the process model is simply an XML file living in the resources of that development project.
- Source code to provide the REST API for clients, which is "normal Java"
- Some Java code to do the customer scoring
- Glue code to implement the REST call to the CRM system.
- Form for the user to approve customer orders

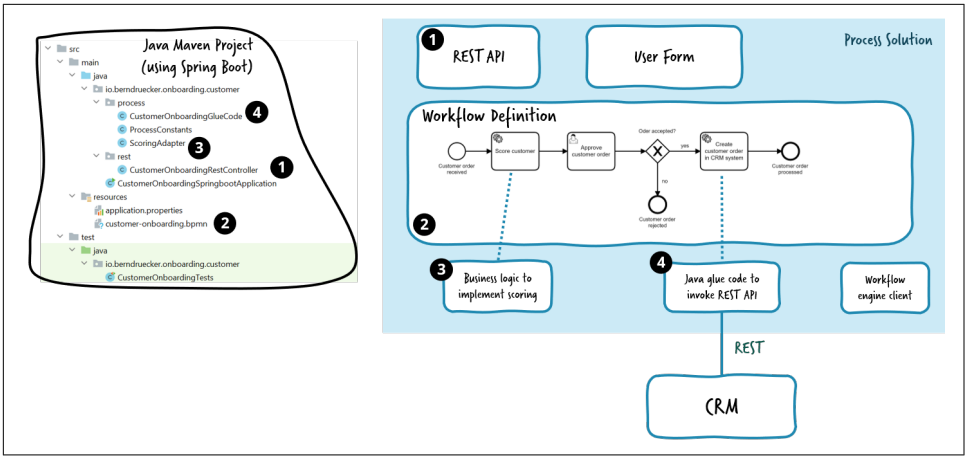


Figure 2-6. A process solution is a development project containing all important artifacts like the process model, glue code and test cases

Let's see how these pieces look when using Camunda Cloud.

First, the process model needs to be deployed to the workflow engine. While you could do that directly via the graphical modeling tool or via the API of the engine, the easiest is to hook in the normal deployment mechanism of your microservice. In this case, this is an auto-deployment during startup of Spring Boot as shown in the following code snippet.

```
@SpringBootApplication
@EnableZeebeClient
@ZeebeDeployment(classPathResources="customer-onboarding.bpmn")
public class CustomerOnboardingSpringbootApplication {
}
```

Now you can use the workflow engine API to create new instances of a process, e.g., when a new REST request is received.

```
@RestController
public class CustomerOnboardingRestController {

  @Autowired
  private ZeebeClient workflowEngineClient;

  @PutMapping("/customer")
  public ResponseEntity onboardCustomer() {
    onboardCustomer();
    return ResponseEntity.status(HttpStatus.ACCEPTED).build();
  }

  public void startCustomerOnboarding() {
    HashMap<String, Object> variables = new HashMap<String, Object>();
    variables.put("automaticProcessing", true);
```

```
        variables.put("someInput", "yeah");

        client.newCreateInstanceCommand()
            .bpmnProcessId("customer-onboarding") //
            .latestVersion() //
            .variables(variables) //
            .send().join();
    }
```

You can find more sophisticated code samples on the website for this book, for example to return a synchronous response in case the onboarding process returns in milliseconds.

On the process model you need to add expression language to implement the decision which path to take in the process model, as shown in Figure 2-7.

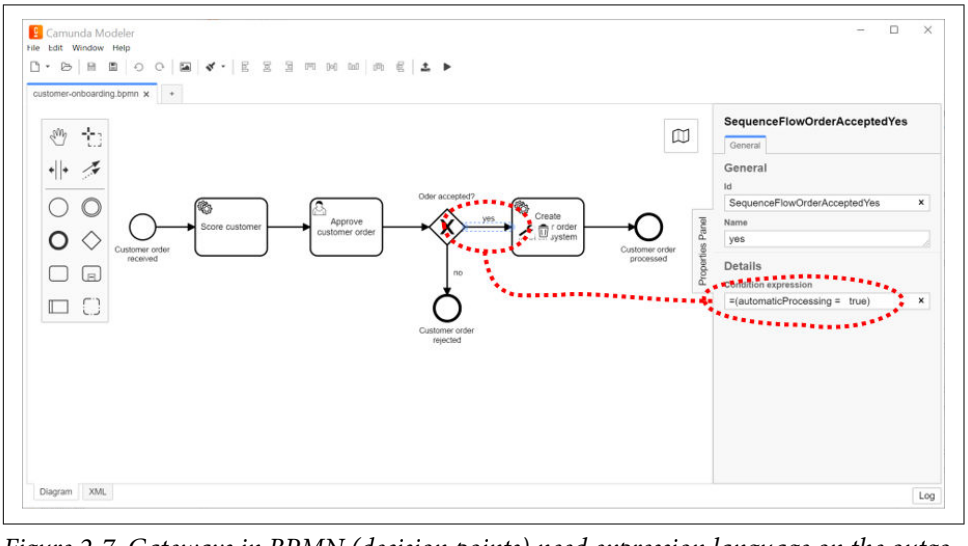


Figure 2-7. Gateways in BPMN (decision points) need expression language on the outgoing sequence flows

Camunda cloud uses FEEL (Friendly Enough Expression Language), which is a business-friendly expression language standardized in the context of decision engines. It will be described in "Orchestrate Decisions" on page 77. In the example, the expression simply checks a process variable `automaticProcessing`. If it is true, the process continues on the "yes" path.

Then you have to define your glue code as shown in the following code snippet:

```
@Component
public class CustomerOnboardingGlueCode {

 @Autowired
 private RestTemplate restTemplate;
```

```
  @ZeebeWorker(type = "addCustomerToCrm")
  public void addCustomerToCrmViaREST(JobClient client, ActivatedJob job) throws IOException {
    log.info("Add customer to CRM via REST [" + job + "]");

    // TODO some real logic to create the request
    restTemplate.put(ENDPOINT, request);
    // TODO some real logic to process the response

    // let the workflow engine know the task is completed
    client.newCompleteCommand(job.getKey()).send().join();
  }
}
```

This code needs to be connected to the process model. In Camunda Cloud this is done by logical task names, as visualized in Figure 2-8.

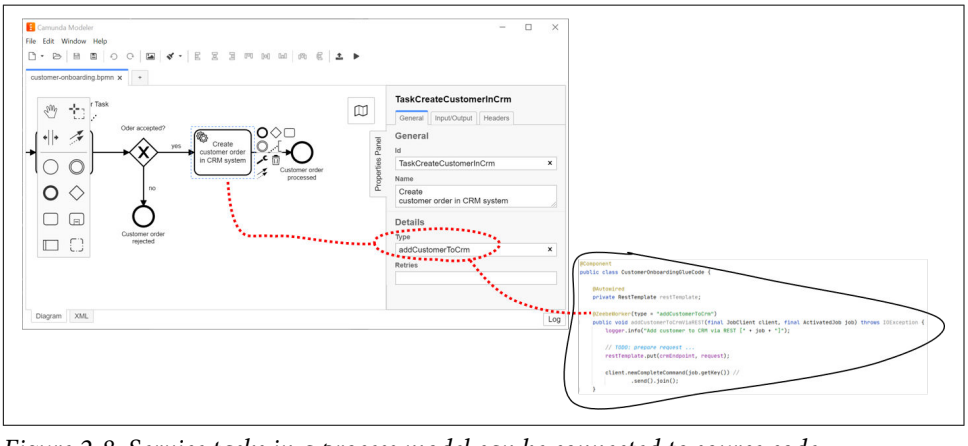


Figure 2-8. Service tasks in a process model can be connected to source code

In order to start the microservice, you need the workflow engine up and running. In case of Camunda Cloud, this means that you will create a new "Zeebe cluster" via the cloud console available on *https://console.cloud.camunda.io/*. Zeebe is the name of the workflow engine fueling Camunda Cloud.

You will receive connection details that you need to add to your application configuration, which in our example is a file called *application.properties*. Spring allows to overwrite these connection details easily, e.g., via environment properties, which will be handy when you want to run the application in a production environment later.

After starting the Java Spring application, you can invoke the REST API with the REST client of your choice, e.g., cURL:

```
curl -X PUT
     -H "Content-Type: application/json"
```

```
-d '{"someVariable":"someValue"}'
http://localhost:8080/customer
```

This will execute the REST code shown above, which in turn starts a new process instance in the workflow engine. A good way to understand the nature of a workflow engine is to look into the operations tooling. Figure 2-9 gives an example, showing the just started process instance and what data are available surrounding it.

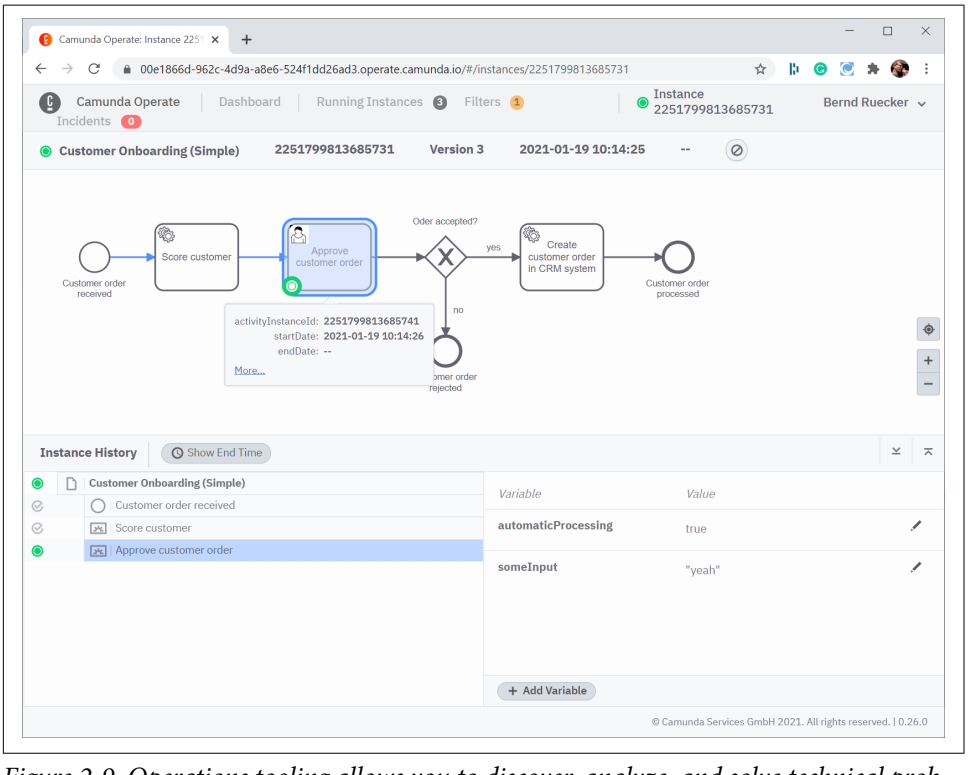


Figure 2-9. Operations tooling allows you to discover, analyze, and solve technical problems related to processes

The process model is source code and implements important parts of the business logic. So it should be tested as other parts of your business logic are. In Java, this means to write unit tests using JUnit. While finishing this book, the assertion API was still in flux, so please check the source code of the example for the latest source code with Camunda Cloud. It will look like the following:

```
@Test
void testHappyPath() throws Exception {
 // simulate an incoming REST call that will kick off a new process instance
 customerOnboardingRest.onboardCustomer();

 // assert that a process was started
```

```
    ProcessInstanceEvent pi = assertProcessInstanceStarted();

    // Assert that a job (pub/sub mechanism of the workflow engine) for scoring was created
    RecordedJob job = assertJob(pi, "scoreCustomer");
    assertEquals("TaskScoreCustomer", job.getBpmnElementId());
    assertEquals("customer-scoring", job.getBpmnProcessId());
    // and complete the task, but executing some fake logic instead of the real adapter
    execute(job, new JobHandler() {
      public handle(JobClient client, ActivatedJob job) {
        // do some fake behavior instead of the real Java code
      }
    });

    // verify that human task was created
    RecordedHumanTask task = assertHumanTask(pi);
    assertEquals("TaskApproveCustomerOrder", task.getBpmnElementId());
    // ... maybe do more assertions ...
    // and simulate it being completed with approval
    Map variables = new HashMap();
    variables.put("automaticProcessing", true);
    complete(task, variables);

    // Assert the next job for the call to the CRM system was created
    job = assertJob(pi, "create");
    assertEquals("TaskCreateCustomerInCrm", job.getBpmnElementId());
    // and trigger its execution with the normal behavior
    execute(job);
    // A mock rest server was injected into the glue code by Spring,
    // so we can verify the right request was sent
    mockRestServer
      .expect(requestTo("http://localhost:8080/crm/customer")) //
      .andExpect(method(HttpMethod.PUT))
      .andRespond(withSuccess("{\"transactionId\": \"12345\"}", MediaType.APPLICATION_JSON));

    assertEnded(pi);
  }
```

The process solution behaves like a typical Java Spring Boot project. You can check it into your normal version control and build it using your normal CI/CD pipeline like any other Java project. For instance, the sources of this example live in GitHub and are continuously built by TravisCI.

The full source code is available online and I can only recommend to play around with it, as this will help you to get a better basic understanding of a workflow engine for the upcoming discussions.

Applications, Processes, Workflow Engines

A typical question is about the relationship between applications, workflow engines, process definitions and process instances.

Assume you use the workflow engine as a service. Then, you can deploy many process definitions on that workflow engine. For every process definition, you can run zero to many process instances. You could also use that workflow engine from many different applications or microservices.

All of this is comparable to a database installation, where you can create multiple tables and connect many different applications to them.

However, it might be advisable to use seperate workflow engines for seperate applications, as this improves isolation. Especially if you embrace microservices, this is typically the way to go as described in "Decentralized Engines" on page 118.

So, for example, the team responsible for order fulfillment might operate a workflow engine. They do not share this with the team doing payments, as they want to be isolated from anything the payment team does. But they connect not only the order fulfillment application, but also the order cancellation application to that engine. Both applications deploy their own process definitions. This example is visualized in Figure 2-10.

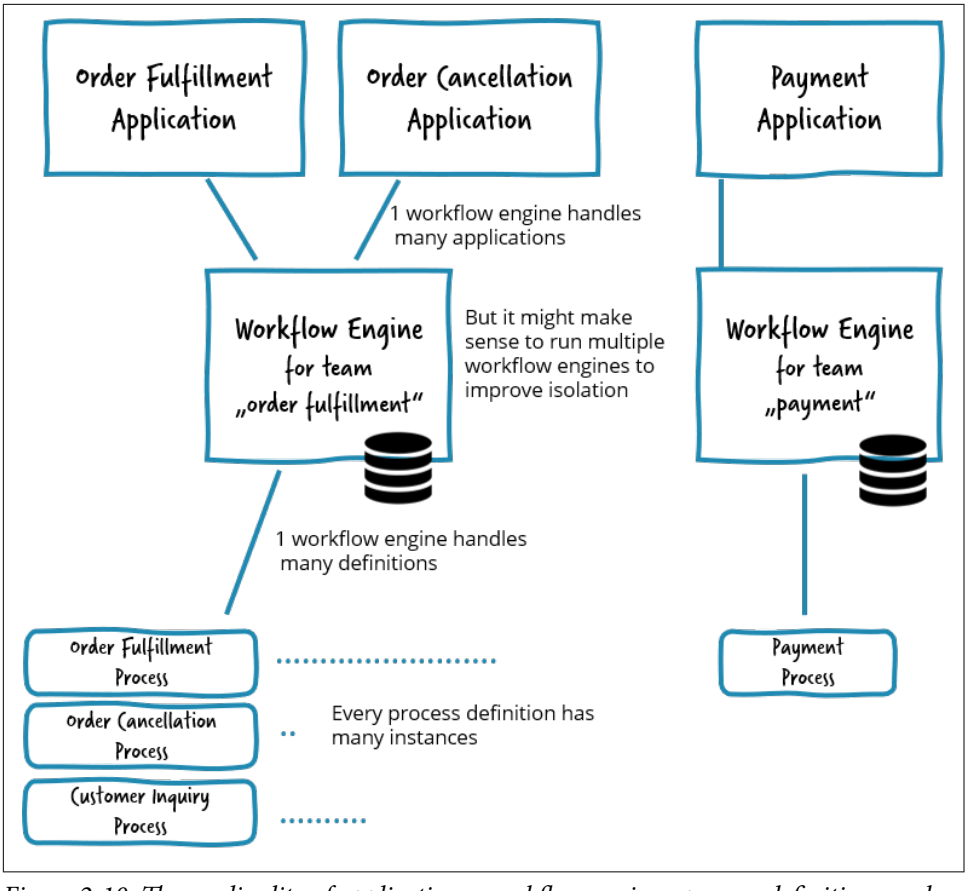


Figure 2-10. The cardinality of applications, workflow engines, process definitions and instances summarized

Typical Workflow Tools in a Project's Life Cycle

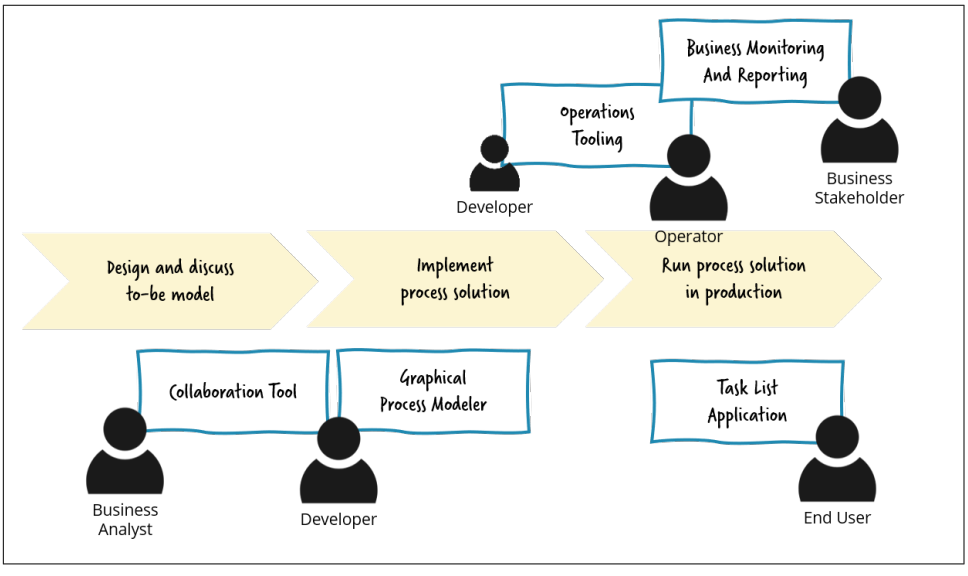


Figure 2-11. Most workflow engines are flanked by other tools that might be valuable at different lifecycle stages of the project, for different stakeholders

Most workflow engines are flanked by tools that help you leverage the full potential of process automation. Figure 2-11 shows a typical stack that might be provided as an integrated platform by your vendor. It provides the following tools:

- Graphical Process Modeler
- Collaboration Tool
- Operations Tooling
- Tasklist Application
- Business Monitoring and Reporting

Let's briefly go over these tools to understand how they are used in process automation projects. Note that good tools allow to unbundle the platform, so that you are not forced to use a big pile of tools, but select what really helps you.

Graphical Process Modeler

A graphical proces modeler allows you to, well, model your processes graphically. Figure 2-12 shows an example.

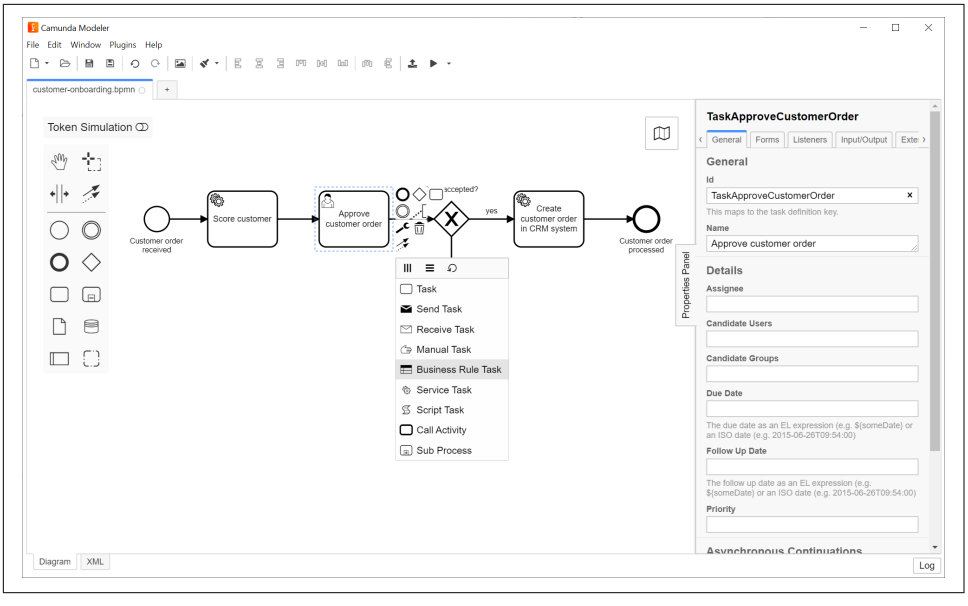


Figure 2-12. Graphical modeling tools allow to edit process definitions

While a graphical modeler might be a valuable tool for business analysts, the focus in this book is on executable processes, so let's treat this modeler as tool for the developer. This is something some tools do a good job with, and some don't. Let's give two examples.

The modeler should be able to work file-based, allowing you to store the process model as part of your version-controlled source code. This makes it easy to keep it in sync with your source code. Some tools force you to use a separate repository, which can make the experience more brittle.

And the modeler should provide easy ways to edit all technical details that are important to make a model executable. This includes referencing glue code and other aspects you saw earlier in this chapter.

Graphical modelers are very handy in process automation projects. Just make sure you select a developer-friendly tool stack, as the wrong tool might easily become an obstacle in software development. Look at how the tool fits into your development environment.

Collaboration Tools

During the initial discussion on how to automate a certain process it is often valuable that different people collaborate on process models. This includes people from many roles, especially business analysts, developers, and methodology or subject matter

experts. So a good example of a useful feature in this phase is to share a diagram with others and let them comment on it as shown in Figure 2-13.

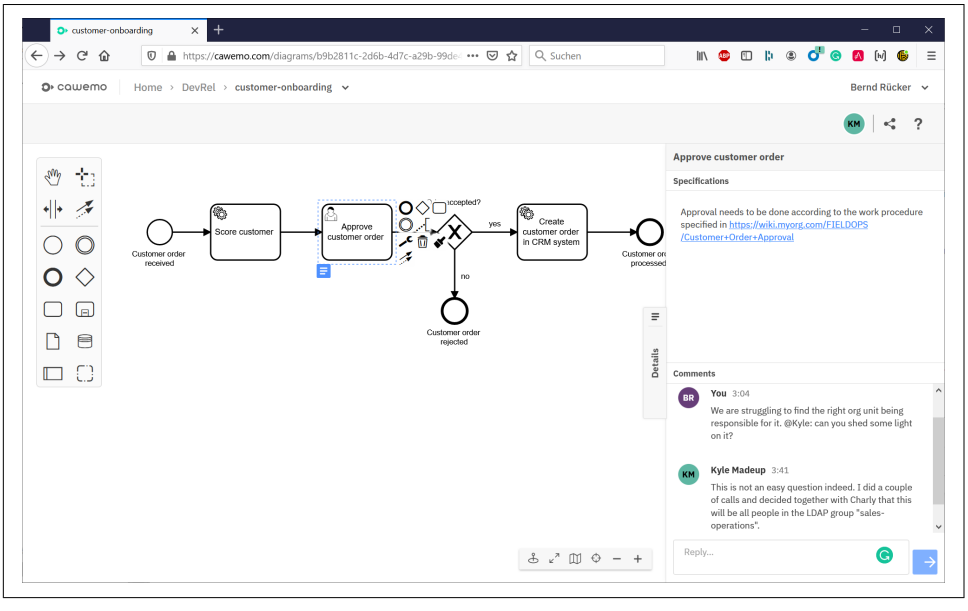


Figure 2-13. Collaboration tools allow you to share and discuss models amongst a big variety of people with different roles

Collaboration tools typically have their own repositories where they store process models. It is important that these models are not treated as source code of the process solution. Instead, the developer keeps looking at the process model stored in their Git to makre sure they can control the process model that is used as source code. This is often a thin line and "The Power of One Joined Model" on page 207 will examine this complex topic in more detail. For now, let's remember that collaboration tools can help us in the discussion around to-be process models, but they are not used to implement the process solution.

Operations Tooling

Whenever you are live with a process, you need a tool that allows you to discover, analyze and solve problems related to processes, like shown in Figure 2-14.

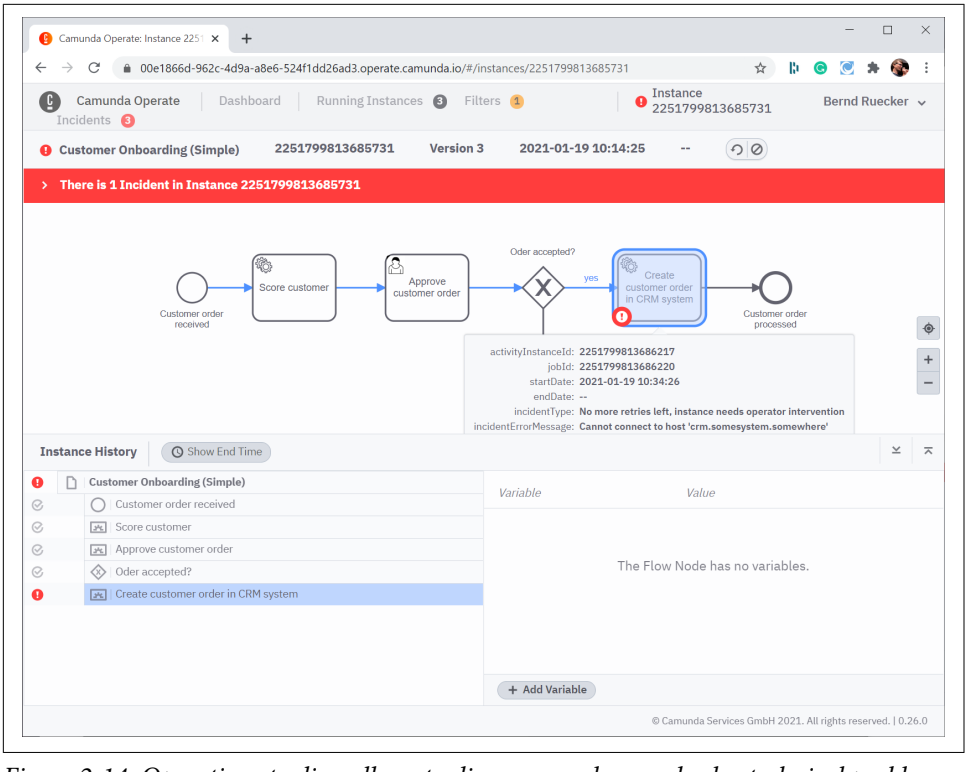


Figure 2-14. Operations tooling allows to discover, analyze and solve technical problems related to processes

Imagine there is a problem with the service call to the CRM system. You first need monitoring that will recognize that problem, e.g., because incidents are piling up. You will also want to send alerts or integrate into your existing APM (Application Performance Monitoring) tool, so the right person in charge gets actively notified. After that alert, the tool needs to support root cause analysis to understand the problem at hand (e.g., some endpoint URL has changed) and then finally allows fixing the issue (e.g., updating a configuration option and trigger a retry), probably also for a big number of affected process instances.

These operations tools might also be used by developers to understand certain incidents in production, while at the same time being used to "play around" during development.

Tasklist Applications

A process model can contain tasks where a human should do something. Now, if a process instance arrives in such a task, the human needs a way to know that it is their

turn. For this use case, most vendors ship a tasklist application like the one shown in Figure 2-15.

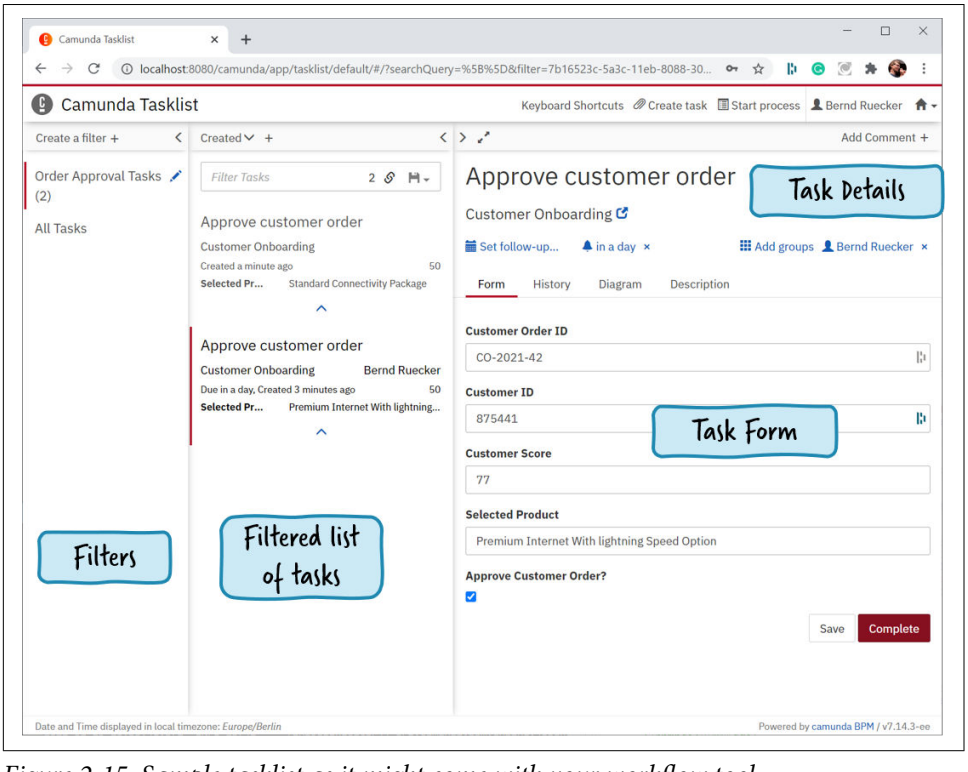


Figure 2-15. Sample tasklist as it might come with your workflow tool

These tools allow end users to see all the tasks they have to do in various running process instances. They can select a task, work on it and let the workflow engine know once they are finished. "The User Interface of User Tasks" on page 85 goes into more details.

Business Monitoring and Reporting

Given your process solution is running in production, your business stakeholders want to monitor processes.

In contrast to operations, these people are much less interested in urgent technical problems, but in the overall performance. For example, these can be measured in cycle times, waiting times, or in-flight business value. They might also want to have some notifications, but typically these are focused on performance indicators. For example, they need to be notified if a process instance is taking too long to process and thus will miss its service level agreement (SLA).

Business stakeholders also care about optimizing the overall process which can be supported by analytics capabilities like a clear view on which process path is often used, which paths are slow, which data conditions often lead to cancellations, and so forth. This information can be derived from the audit data that a worklow engine stores when executing process instances.

Figure 2-16 shows an example dashboard including different kind of data points around a hiring process.

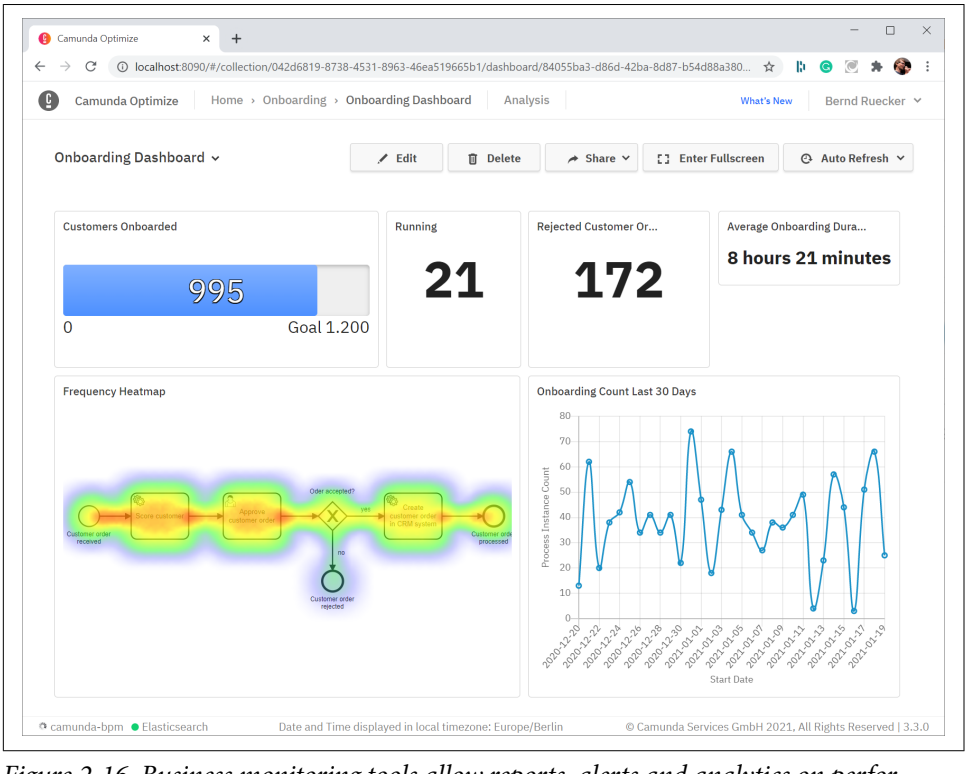


Figure 2-16. Business monitoring tools allow reports, alerts and analytics on performance indicators

Conclusion

This chapter described a workflow engine and process automation platforms in more detail. It dived into a first hands-on, executable example that should help you understand process solutions and workflow engines.

This equips you for diving into more details around developing process solutions in the next chapter.