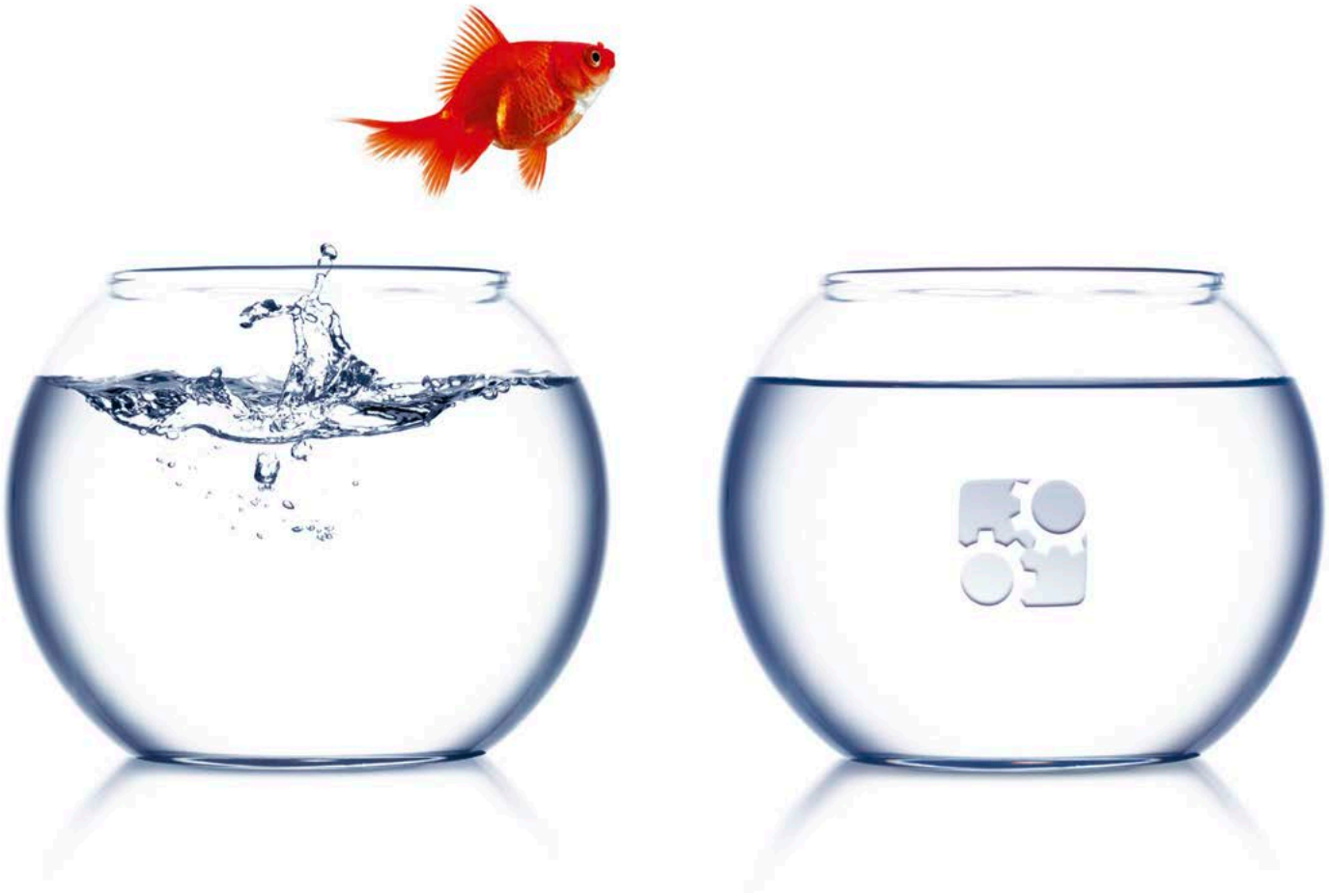
camunda
the business process company

How to migrate processes from existing solutions to Camunda BPM



camunda.com

Contents

Introduction

Most customers don't start on a green field but migrate existing solutions to BPMN 2.0 processes in Camunda BPM. Typical examples for such legacy are:

- Third party process engines
- Hard coded applications
- Clumsy constructs consisting of code, database tables, triggers and magic scripts

Migration is the most important step when going live – this whitepaper summarizes our experience in customer projects.

Migrate process definitions

Processes are deployed as valid BPMN 2.0 files in Camunda BPM. Therefore, the processes have to be deducted from the existing solution.

Our recommended approach is to re-model the existing processes. This is actually what we've done in almost every project so far, as the number of process definitions is usually limited and the time you need to re-model them is much less than the time you would need to create and test automated conversion. You can also leverage the full power of BPMN 2.0 – improving the results dramatically.

Depending on your situation, there may be other options available, though we usually do not recommend them:

1. The old engine is also a BPMN 2.0 process engine. So, in theory, there is no need to adjust the process models. Unfortunately this isn't true in real-life as you always have to change some details such as extensions used by the vendors. Often other solutions also have worse BPMN coverage, so the process model contains workarounds which are no longer necessary. It's a pretty good idea to get rid of them.
2. There is an XML process model in a different language (such as BPEL or XPDL) or some proprietary format. In that case we always discuss automated conversion into BPMN. However, we usually advise against any automatism – as this is really hard to do, which often means that it's a larger effort to create the conversion tool than it would be to re-model the processes. The resulting process models also can't leverage the BPMN possibilities or comply to typical best practices. So, unless you have to migrate thousands of process models and don't care about the beauty of the result we would not recommend this approach.

Migrate process instances

Migrating the process instances is unavoidable if you have long-running processes – as some of them are running at any given time. There are two basic approaches:

1. **Concurrent Operation:** You keep operating the old solution until no process instances are left there. In the meanwhile you start new process instances on Camunda BPM. To do this you have to implement a kind of switch to route incoming requests to the old or new system. Bear in mind that you need to take care of operating both systems in parallel, e.g. checking failures, calculating KPI's or performing instance counts.

   PRO: You do not need to migrate running process instances and you save effort on this. Also, going live with Camunda BPM is typically less risky as the old system is still in place and operating.

   CON: Operating two systems in parallel for a longer time is additional effort and the switching logic normally is not easy to implement and test for all corner cases. Also, as everybody is normally eager to throw out the old stuff it might even be a downer for motivation.

2. **Big Bang Migration:** You stop the old system and make sure all process instances have reached a wait state. Then you migrate all instances to Camunda BPM and fire up the new system. For that, you need to map all possible wait states from the old processes to the BPMN process definition, read the data from the old system and run a "migration script" to create the process instances in Camunda BPM in the correct state.

   PRO: After the migration you only have one system to operate and you can test the migration perfectly beforehand to avoid any surprises.

   CON: Implementing and testing the migration script is effort you have to do. Also, usually the run time of the migration scripts and possible additional health checks afterwards enforce some downtime.

In real-life both approaches are used. There is no clear recommendation, so you have to decide yourself based on your exact situation.

Concurrent operation is completely specific to your architecture and technology. We are pretty happy to help you in terms of consulting – but there is no clear blueprint of how to achieve this.

In contrast, the *big bang migration* always involves a "migration script", which requires some knowledge of Camunda BPM for implementation. This is described in the remaining whitepaper, please note that sample code is available.

Sample Code Link on GitHub: camunda.com/whitepaper/migration

Big Bang Migration

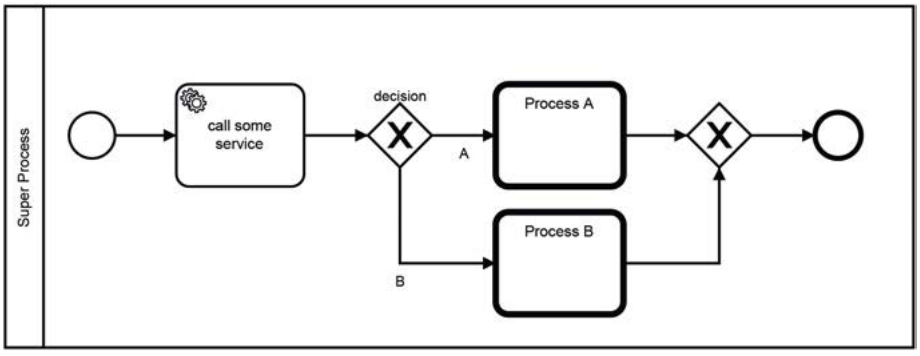
Use Migration Scenarios to create process instances in the correct state

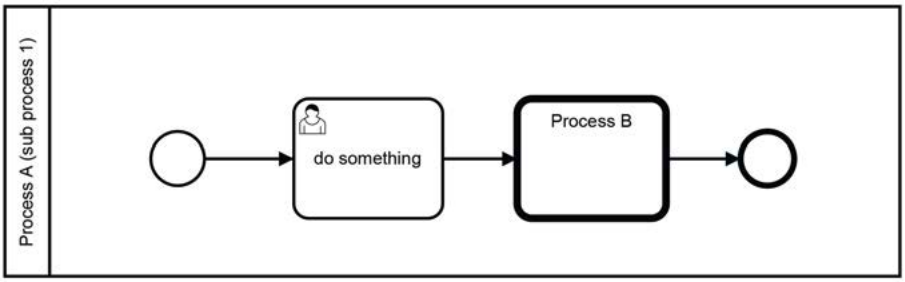
The goal sounds rather simple: You want to create process instances in a state which might not easily be reached via normal ways in the process model.

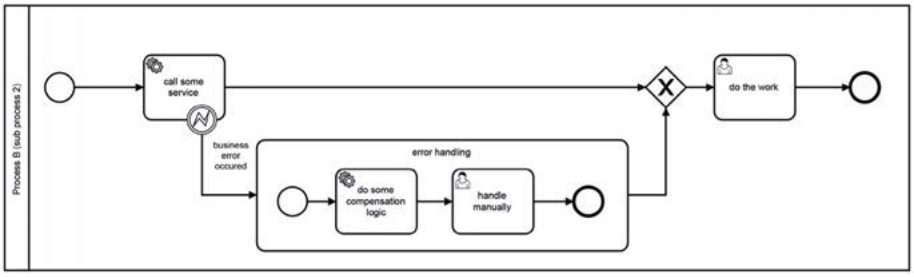
However, in order to do so you do not want to trigger any service tasks or wait for human interaction on the way. Also, the desired state might even be somewhere down a hierarchy of sub-processes. This makes the task a bit more complex.

The recommended way of doing this is to create an adjusted version of the process definition, containing own elements for what we call “migration scenarios”. This means that we really add elements to the process model, which then allows us to basically leverage the normal BPMN 2.0 execution in order to steer process instances to the correct state.

Lets start with the example shown in figure below.







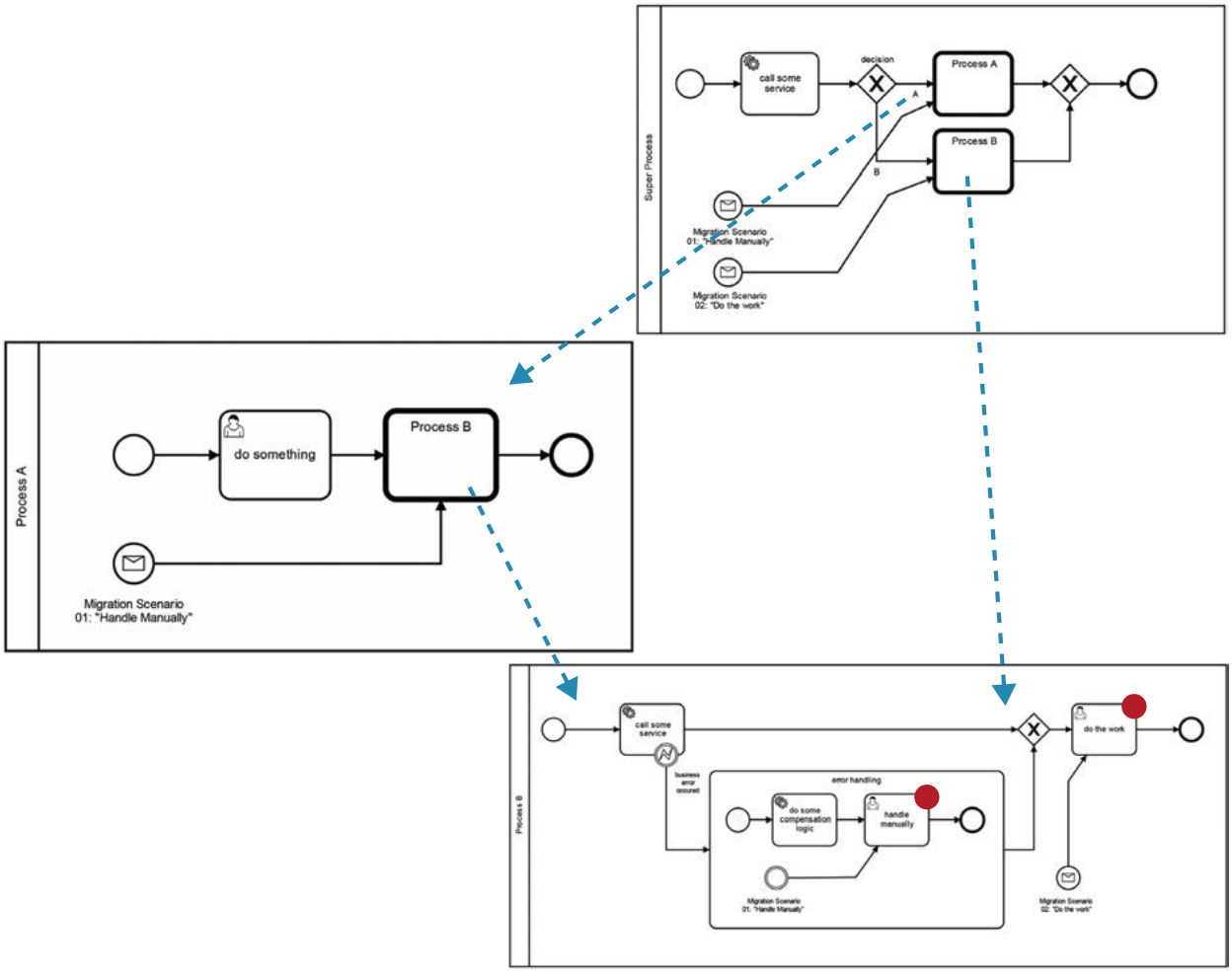
The two red dots mark the places to which we want to migrate. We defined two scenarios here:

- **Migration Scenario 01:** jump into the User Task “handle manually”, which is part of a sub-process and a call hierarchy.
- **Migration Scenario 02:** jump into the User Task “do the work” down the call hierarchy.

We adjusted the process model to include the two scenarios. To be precise, we added:

- Message Start Events with a special naming convention to start process instances at a different activity (please note that no vendor-specific workaround is necessary to do this)
- An Intermediate None Event as starting point in the sub-process. This is actually a workaround because BPMN cannot directly jump into a node within a sub-process
- An extension to the Call Activities to configure which migration scenario should be triggered in the sub-process.

The resulting process model is shown in figure below.

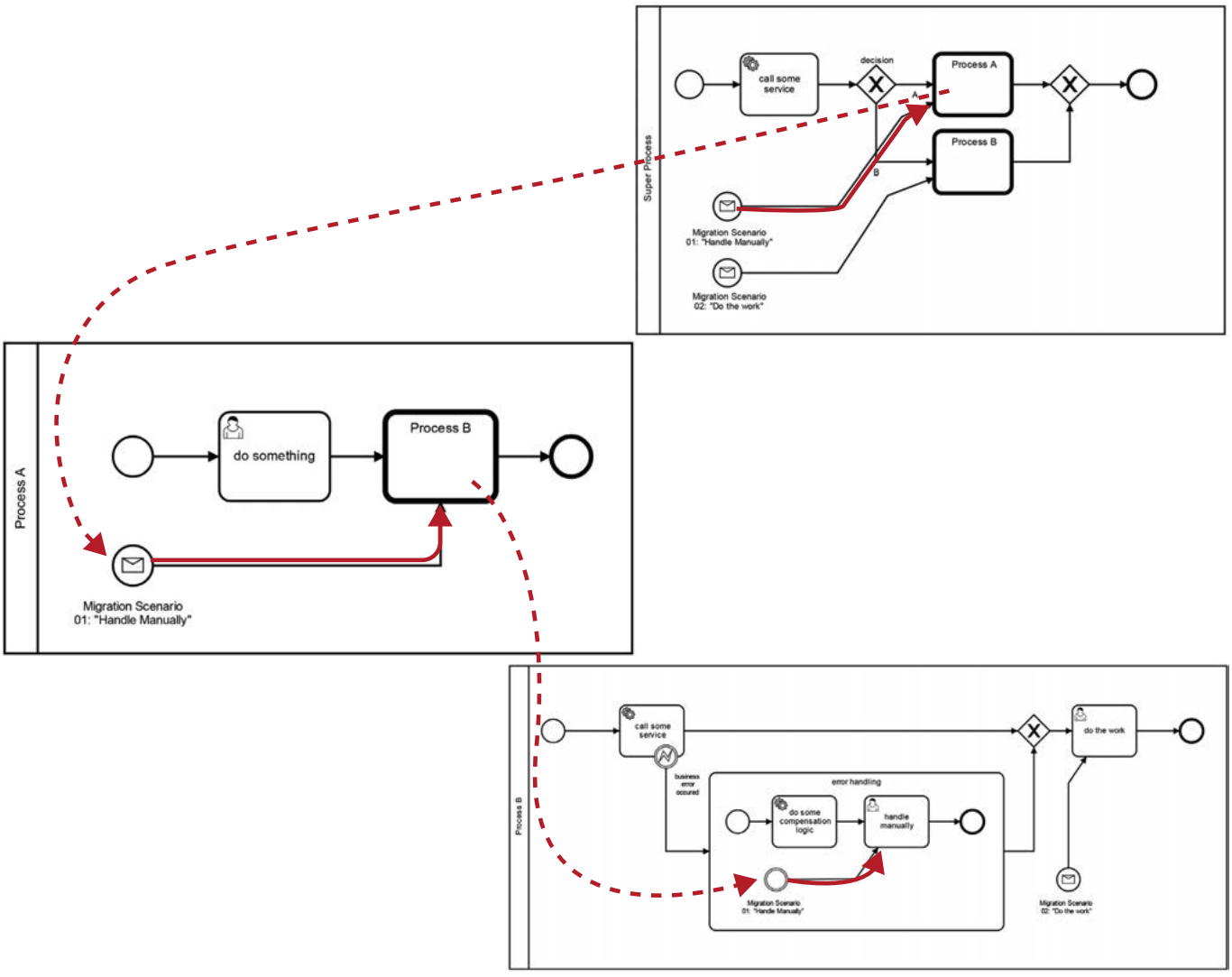


Now we can start process instances by using the normal public Camunda API (Java or REST). The Java way, for example, looks like this:

```
Map<String, Object> variables = new HashMap<String, Object>();
variables.put("migrationScenario", "01");

runtimeService.startProcessInstanceByMessage("migration-example-super-process#MIGRATION_
SCENARIO_01", variables);
```

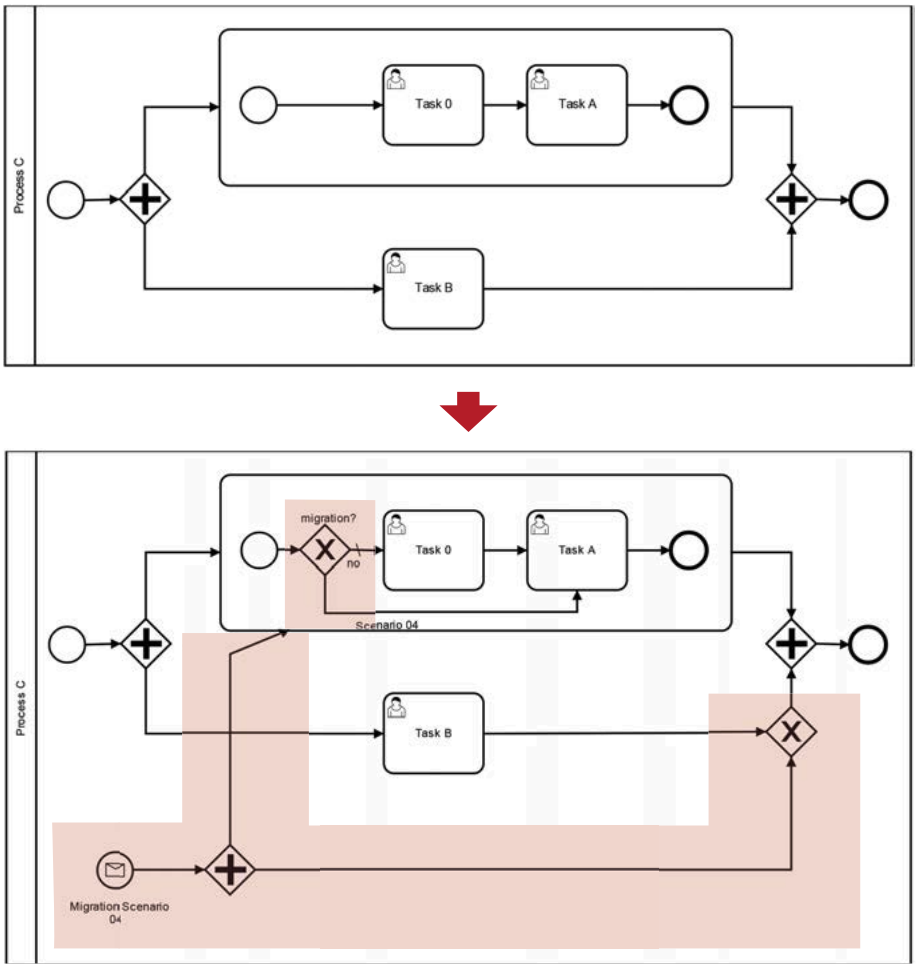
Note that no tasks are executed on the way to the desired state, which means that no unwanted side effects occur. Now you can easily write a migration script which triggers the right process instances with the right data via the Camunda API. In Scenario 1 for example, the “path” illustrated in figure below is taken:



Of course you can deploy the process model without the migration scenarios as version 2 immediately after the script runs. Then, freshly started process instances don’t have any overhead of migration scenarios.

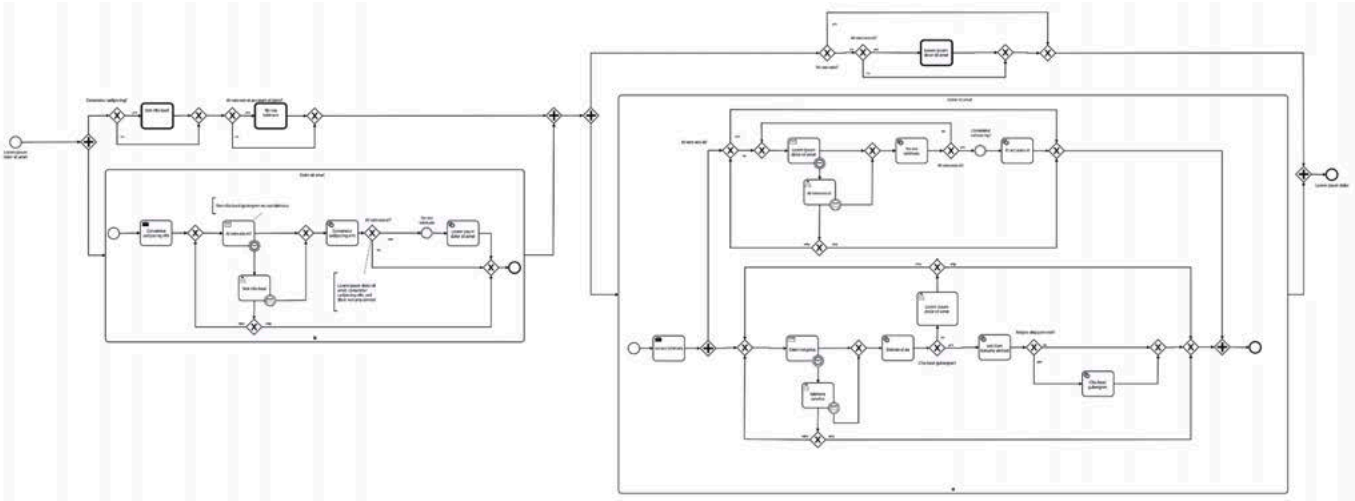
Adjust your process models

Our approach tries to limit vendor-specific extensions (or hacks) to a minimum. As BPMN cannot “jump” into scopes or parallel paths there are situations which cannot be handled out-of-the-box by the migration scenarios. In that case, you have to adjust the process models. We want to sketch one typical example shown here in figure below.



The migration scenario defines that we want to end up in Task A – and only Task A (so Task B has already been completed). In order to jump into the parallel paths, we need to have an additional parallel gateway for the migration and a decision within the sub-process to bypass “Task 0” – as we cannot directly go into the sub-process on Task A. The changes are marked in red.

This works pretty well. As an example, figure below shows the structure of a real-life process to which we applied the migration.



Summary and possible alternatives

The sketched approach works really well in our real-life projects. However, there are some limitations that we want to bring to your attention:

- No history: Historic data such as the audit trail of the current process instance up to the current state is not migrated. However, normally it is still available in the old system and maybe in your Data Warehouse or Business Intelligence solution – as a lot of customers load the historic data into these systems for monitoring and reporting purposes.
- Not all scenarios are possible: As already discussed, you sometimes have to adjust the process models. However, these are usually rather minor changes.
- Java Migration: This approach actually requires to run Java code for the migration. As we do see this as an advantage because you have all logic and validation of the process engine in place, it means that you cannot just run some magic database script for the migration.

However, there are some major advantages of this approach:

- Clarity and visibility: As you modeled the migration scenarios, they are visible in the history data and can be visualized in cockpit. Whenever you analyze a process instance later on, you will perfectly see that this was migrated and what migration scenario was used.
- Simplicity: The solution is rather easy to understand, easy to implement and the changes in the model are not complex. Even if you have to put some effort in modeling the migration scenarios – you have to think them through either way, so why not document them in the same step!
- No magic: We use the public API and leverage the BPMN process engine. That minimizes the risk of breaking something.

Possible alternatives would have been:

- Create a migration script which steers the process through the process model with the public API until it reaches the desired state in the “normal” way. This is rather hard, as you have to limit side effects at the same time, meaning foremost that you cannot call external services! However, you are still dependent on data, for example for decisions. This data might change throughout the process instance life, making it really hard to know which value some data had back then when you made that decision. So how do you know how to make the decision in migration?
- Create the process instance directly in the database with the correct state, after all it just means a few rows in the database. This is basically possible, but really hard to implement, as you have to duplicate a lot of logic the process engine normally does. Also, you maybe have to take care that some listeners are called correctly when entering that state, for example sending an email when a User Task was created. However, it might still be doable for simple cases. The biggest downside is that you cannot easily see anything about the migration in the history or in cockpit. There the process instance magically appeared out of nothing. This might even crash some reports you want to do.

Source code

The whole example shown above is available as running example code. Details on the implementation are described in the according project website on GitHub.

Sample Code Link on GitHub: camunda.com/whitepaper/migration

Conclusion

Big Bang migrations are often preferred over concurrent operation of the old solution and Camunda BPM. For Big Bang migrations, you have to re-model your processes in proper BPMN 2.0 and afterwards migrate all existing process instances into the right state in Camunda BPM. This is doable with the sketched Migration Scenarios, which are documented in BPMN at the same time. Some situations require adjusting of the process model – but we see this as the best trade-off to do. The migration itself can be steered via the public Camunda API, e.g. REST, so the biggest issue to solve is to get the input data from your old solution.

Imprint

Europe / Asia

Camunda Services GmbH
Zossener Str. 55
10961 Berlin
Germany

Phone: +49 (0) 30 664 04 09 - 00
E-Mail: info@camunda.com
www.camunda.de

America

Camunda Inc.
44 Montgomery St, Suite 400
San Francisco, CA 94104
USA

Phone: +1.415.548.0166
E-Mail: info@camunda.com
www.camunda.com