Introduction

Table of Contents

Motivation

Motivation

In the last five years, we built custom Camunda task lists for different customers about six times. Most of them were built based on Single Page Application (SPA) technologies, but some were using server-side rendered views. It turned out that some of the issues occurred every time during the implementation.

These were:

- coping with performance issues of the `TaskService` by the big amount of tasks available
- creating high-performance custom queries for pre-loading process variables for tasks
- creating high-performance custom queries to pre-load business data associated with the process instance
- high-performance re-ordering (sorting) of user tasks
- high-performance retrieving a list of tasks from several process engines
- repetitive queries with same result
- creating an archive view for business data items handled during the process execution
- creating an audit log of changes performed on business data items

Many of those issues have to do with the fact that data on single task is written only few times, but is read many times (depending on the user count). For systems with a big amount of users this becomes a serious performance issue if not addressed. One of the possible solutions to most of those issues listed above is to create a special component, which has a read-optimized representation of tasks and is pre-loads tasks from the `TaskService`. In doing so, it decouples from the `TaskService` by the costs of loosing the strong consistency (and working with eventual-consistent task list), but allows for serving a high amount of queries without any performance impact to the process engine itself.

The goal of this project is to provide such component as a library, to be used in the integration layer between the Camunda BPM engine and the task list application.

Features

Features

- mirroring tasks: provides a list of tasks in the system including all task attributes provided by Camunda BPM Engine

- reacts on all task life cycle events fired by the process engine

- high performance queries: creates of read-optimized projections including task-, process- and business data

- centralized task list: running several Camunda BPM Engines in several applications is standard for larger companies. From the user’s perspective, it is not feasible to login to several task lists and check for relevant user tasks. The demand for the centralized task list arises and can be addressed by `camunda-bpm-taskpool` if the tasks from several process engines are collected and transmitted over the network.

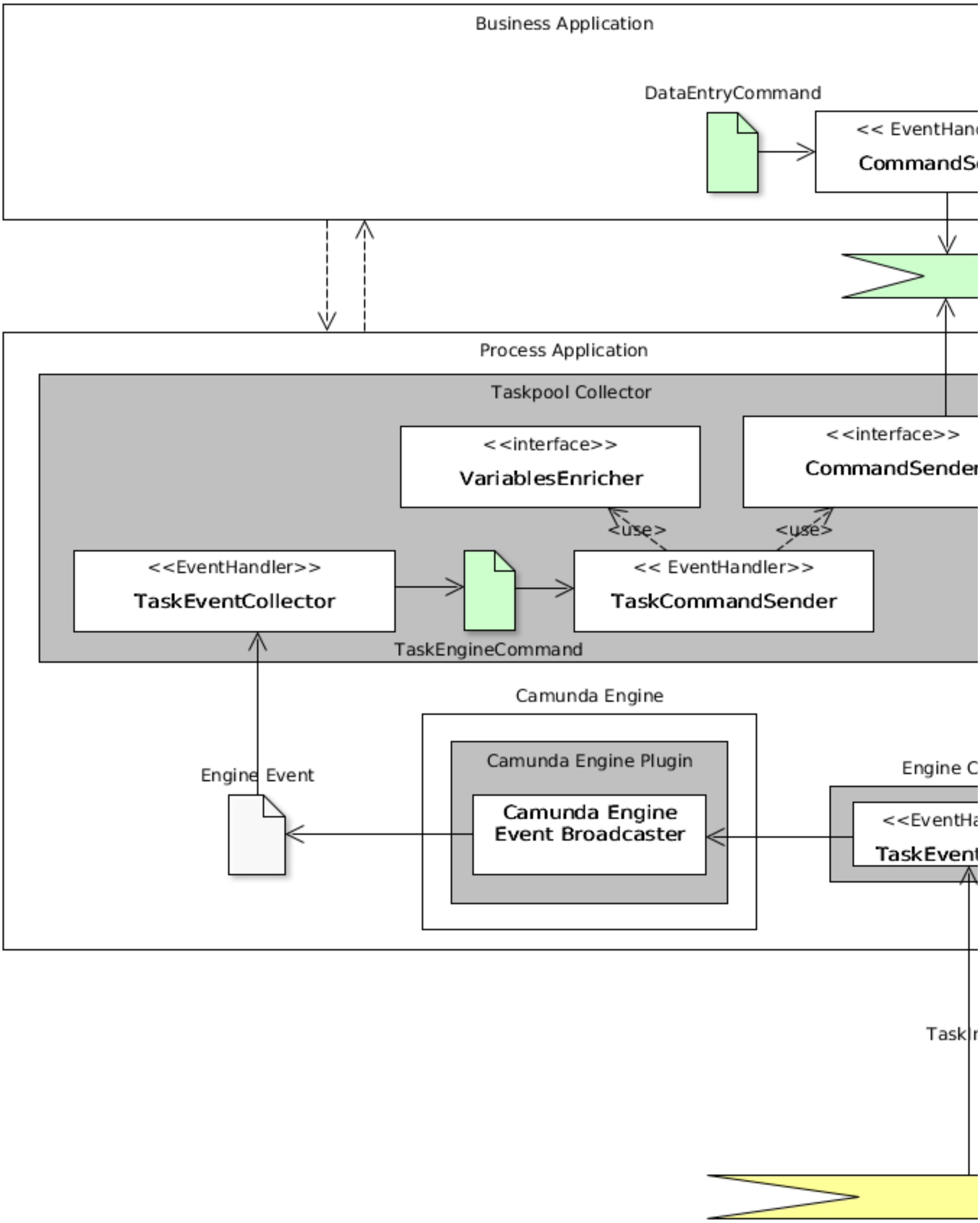
- data enrichment: all use cases in which the data is not stored in the process result in a cascade of queries executed after the task fetch. The task itself has only the information of the `executionId`, so you have to query the `RuntimeService` for the execution, load some variables from it and query external systems for further values. Another approach is presented in the post from Jan Galinski Process business data with JPA, but still results in a query on the task fetch. In contrast to that, the usage of the `camunda-bpm-taskpool` with a data enrichment plugin mechanism (allowing to plug-in some data enricher on task creation) would allow for caching the additional business data along with the task information, instead of querying it during task fetch.

Solution Idea

Solution Idea

The solution is implementing the Command Query Responsibility Segregation (CQRS) pattern, by collecting the tasks from the process engines and creating a read-optimized projection with tasks and correlated business data events. In doing so, `camunda-bpm-taskpool` provides several independent components (see below) which can be deployed in different scenarios (see below). The library is implemented using Kotlin programming language and relies on SpringBoot as execution environment. It makes a massive use of Axon Framework as a basis of the CQRS implementation.

The following diagram depicts the overall architecture.


Business Application
DataEntryCommand
<< EventHan
CommandS
Process Application
Taskpool Collector
<<interface>>
VariablesEnricher
<<interface>>
CommandSender
<use>
<use>
<<EventHandler>>
TaskEventCollector
<< EventHandler>>
TaskCommandSender
TaskEngineCommand
Camunda Engine
Camunda Engine Plugin
Camunda Engine Event Broadcaster
Engine Event
Engine C
<<EventHa
TaskEvent
Task

Further outlook

Further outlook

This library serves as a foundation of several follow-up projects and tools:

- Skill-based-routing: based on information stored in the taskpool, a skill-based routing for task assignment can be implemented.

- Workload management: apart from the operative task management, the workload management is addressing issues like dynamic task assignment, optimal task distribution, assignment based on presence etc. For doing so, a task pool to apply all these rules dynamically is required and the `camunda-bpm-taskpool` component can be used for that.