Introduction

Table of Contents

Motivation

Motivation

In the last five years, we built different process applications on behalf of the customer several 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

Task List

A task list is a application allowing to represent a list of user tasks. This list is individually created based on user's profile (including authorizations based on roles). The following features are provided by `camunda-bpm-taskpool` library:

- user task API providing attributes important for processing
- 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 a common use case 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. 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.

Archive List

An archive list provides a list of business objects processed during the execution of business process. Such a business object lifecycle spans over a longer period of time than the process instance - a common requirement is to get a list of such objects with different statuses like preliminary, in process or completed. The `datapool` library provides the following features:

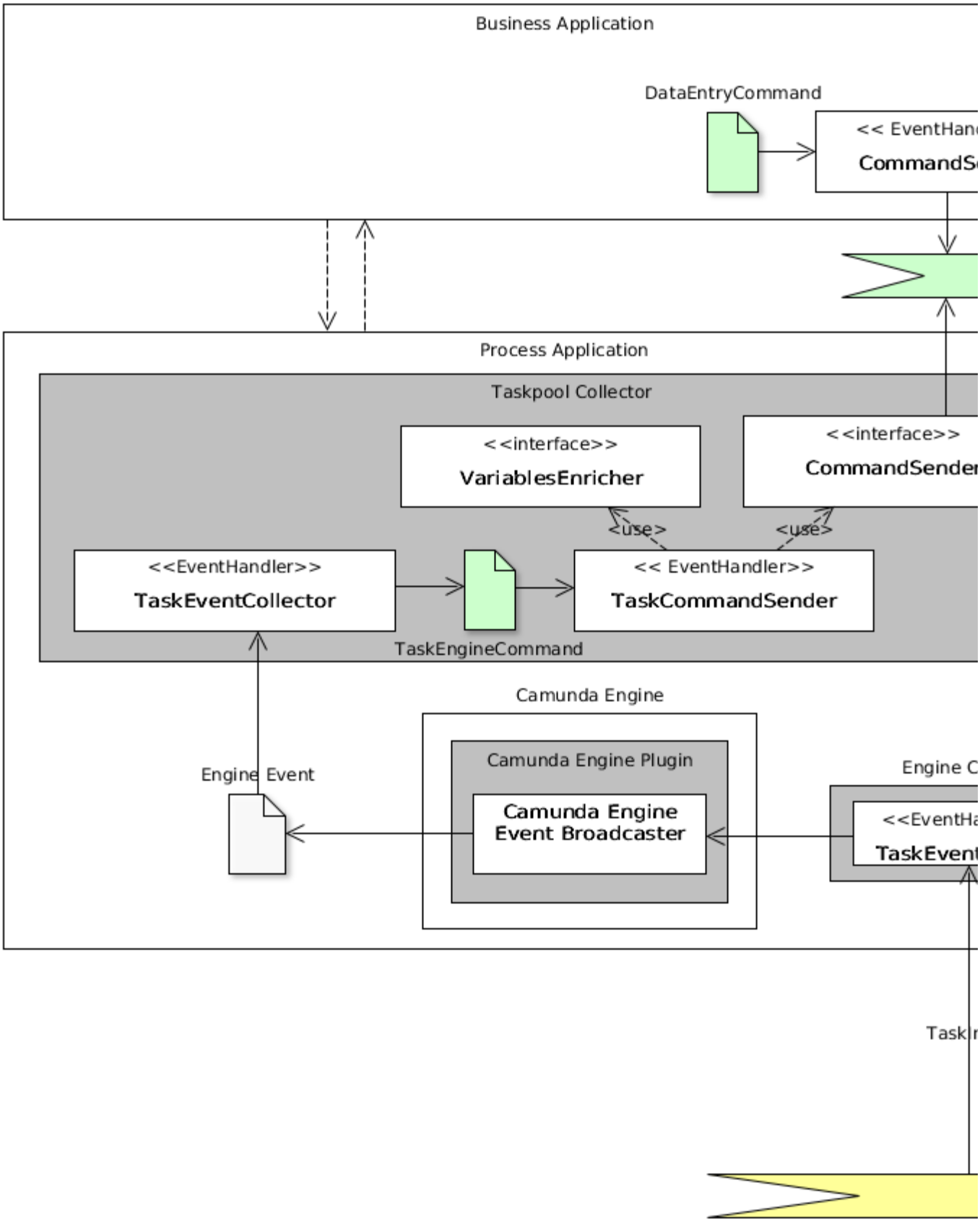
- business object API providing attributes important for processing
- business object modification API for creating an Audit Log
- authorization API for business objects

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
Taskl

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.

Quick Start

Quick Start

Start example task list and example process application

To start quickly, just start the Example Applications. For doing so, please consult the Example Section.

Configure your existing process application

Apart from the example application, you might be interested in integrating task pool into your existing application. To do so, you need to enable your Camunda process engine to use the library. For doing so, add the `camunda-bpm-taskpool-engine-springboot-starter` library. In Maven, add the following dependency to your `pom.xml` :

```
<dependency>
  <groupId>io.holunda.taskpool</groupId>
  <artifactId>camunda-bpm-taskpool-engine-springboot-starter</artifactId>
  <version>${camunda-bpm-taskpool.version}</version>
</dependency>
```

Now, find your SpringBoot application class and add an additional annotation to it:

```
@SpringBootApplication
@EnableTaskpoolEngineSupport
public class MyApplication {

  public static void main(String... args) {
    SpringApplication.run(MyApplication.class, args);
  }
}
```

Finally, add the following block to your `application.yml`:

```
camunda:
  bpm:
    default-serialization-format: application/json
    history-level: full
  taskpool:
    collector:
      tasklist-url: http://localhost:8081/tasklist/
      process:
        enabled: true
      enricher:
        application-name: ${spring.application.name}  # default
        type: processVariables
      sender:
        enabled: true
        type: tx
    dataentry:
      sender:
        enabled: true
        type: simple
        applicationName: ${spring.application.name}  # default
    form-url-resolver:
      defaultTaskTemplate:  "/tasks/${formKey}/${id}?userId=%userId%"
      defaultApplicationTemplate: "http://localhost:${server.port}/${applicatioName}"
      defaultProcessTemplate: "/${formKey}?userId=%userId%"
```

Now, start your process engine. If you run into a user task, you should see on the console how this is passed to task pool. In order to check the result, just open http://localhost:8081/tasklist/ in your browser.

For more details on configuration of different options, please consult the Working Example and the User Guide.